

# Manual de usuario del Generador Estocástico de series sintéticas GAMWGEN

Alessio Bocco (boccoalessio@gmail.com)  
Daniel Bonhaure (danielbonhaure@gmail.com)  
Guillermo Podestá (gpodesta@rsmas.miami.edu)

09 de October de 2020

## Abstract

Este paquete contiene la implementación de un generador estocástico diario y multi-sitio de series meteorológicas sintéticas. La generación de series sintéticas es un insumo básico para el análisis probabilista de las sequías y sus impactos en el sector agrícola. El generador desarrollado es muy flexible y capaz de generar secuencias de valores diarios de precipitación y temperaturas máxima y mínima. A partir de estas últimas se pueden derivar otras variables como la radiación solar y la evapotranspiración.

El generador tiene dos variantes, una permite generar series para localidades puntuales, generador local, y otra que permite la generación en grillas regulares o en localidades arbitrarias, generador espacial. Esto la de la posibilidad al usuario de generar productos a medida para distintas aplicaciones. Además, el generador es capaz de utilizar variables auxiliares que producen simulaciones condicionadas para su uso en conjunto con modelos de cambio climático o de pronósticos estacionales para evaluar impactos en el largo o corto plazo, respectivamente.

Esta flexibilidad se sustenta en el uso de modelos generalizados aditivos (GAM) que permiten modelar con mucha precisión el comportamiento de las variables meteorológicas y capturar las propiedades estadísticas de los datos observados. La modelación de las variables meteorológicas se divide en dos: por un lado, la ocurrencia y monto de precipitación y por otro, las temperaturas máxima y mínima. La ocurrencia de precipitación se modela a través de un modelo probit mientras que al monto se lo hace a través de una distribución aleatoria Gamma. Para la temperatura se utiliza un modelo autorregresivo condicionado por la ocurrencia de lluvia. A su vez, estos modelos pueden ser espacialmente correlacionados con campos aleatorios Gaussianos que contemplan la variabilidad espacial y temporal regional.

Además del generador, se incluyen diagnósticos estadísticos y gráficos para verificar la bondad de ajuste estadística de los GAM y validar que las series sintéticas sean consistentes con los registros históricos. Los diagnósticos son exhaustivos e incluyen todas las propiedades de las series que podrían afectar el desempeño de las mismas durante el análisis probabilista.

# Contents

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Fundamento</b>	<b>4</b>
<b>3</b>	<b>Tipos de series sintéticas</b>	<b>5</b>
<b>4</b>	<b>Metodología</b>	<b>8</b>
4.1	Introducción a los Modelos Aditivos Generalizados . . . . .	8
4.2	Modelación . . . . .	16
4.2.1	Clima local: Ocurrencia de lluvia . . . . .	17
4.2.2	Clima local: Montos diarios de lluvia . . . . .	18
4.2.3	Clima local: Temperatura . . . . .	19
4.2.4	Tiempo local: Para una estación . . . . .	20
4.2.5	Tiempo local: Para una grilla . . . . .	22
<b>5</b>	<b>Aplicación</b>	<b>24</b>
5.1	Instalar paquetes necesarios . . . . .	24
5.2	Creación de directorios . . . . .	24
5.3	Generación de series sintéticas para una sola estación meteorológica . . . . .	25
5.3.1	Crear archivos de entrada . . . . .	25
5.3.2	Series sintéticas estacionarias . . . . .	28
5.3.3	Series sintéticas pseudohistóricas . . . . .	46
5.3.4	Series sintéticas correlacionadas espacialmente . . . . .	63
5.4	Generación de series sintéticas para una grilla regular . . . . .	85
<b>References</b>		<b>115</b>

## **1 Introducción**

- <sup>2</sup> El análisis de series climáticas es siempre un desafío y más aún en regiones donde la disponibilidad de las mismas es escasa. Este déficit es un problema importante para la realización

4 de la caracterización del riesgo de sequías lo que motivó al SISSA para desarrollar una gen-  
5 erador de datos climáticos. Los generadores estocásticos de clima producen series diarias de  
6 variables climáticas con propiedades estadísticas consistentes a las de los datos observados.  
7 Estas series sintéticas son un insumo muy importante para el análisis cuantitativo del riesgo  
8 de desastres naturales. Este tipo de aplicaciones requiere de largas series temporales para  
9 poder capturar la variabilidad natural del clima de una región. Las principales variables  
10 de importancia son temperaturas máxima y mínima y precipitación diaria. En muchas re-  
11 giones estos datos se encuentran incompletos o son directamente inexistentes. Los registros  
12 suelen tener una duración insuficiente o sólo estar disponibles agregados mensualmente. La  
13 cobertura espacial es otro de los problemas más comunes ya que, en general, las redes de  
14 observación meteorológica son poco densas aún en zonas donde la información meteorológica  
15 es de vital importancia.

16 La mayoría de los enfoques tradicionales para la generación de series estocásticas están lim-  
17 itados por su capacidad de generar datos solamente en localidades para las que se cuenta  
18 con observaciones (por ejemplo, donde existen estaciones meteorológicas). Otra desventaja  
19 de algunas de estas herramientas (sobre todo los generadores no paramétricos basados en  
20 remuestreo de observaciones) es que solamente pueden producir valores dentro del rango  
21 observado en el registro histórico. En este proyecto, el generador desarrollado se basó en el  
22 modelo diseñado por Verdin *et al.* (2016). Este generador estocástico diario multivariado  
23 produce series sintéticas de precipitación y temperaturas máxima y mínima. El generador de  
24 Verdin *et al.* (2016) utiliza cuatro modelos estadísticos para modelar la ocurrencia y montos  
25 de precipitación y temperatura máxima y mínima basados en Modelos Lineales Generalizados  
26 (GLMs). Estos modelos son paramétricos y utilizan regresiones lineales entre las variables  
27 para modelarlas. El proceso de ocurrencia de precipitación se modela como una regresión  
28 probit mientras que los montos se ajustan a una distribución aleatoria Gamma. Las tem-  
29 peraturas máximas y mínimas son modeladas como variables autorregresivas condicionadas  
30 por la precipitación. El generador produce series espacialmente correlacionadas tanto para

31 estaciones individuales como en grillas regulares. Una de las limitaciones fundamentales del  
32 generador de Verdin *et al.* (2016) es que las temperaturas máxima y mínima se generan  
33 en forma independiente para cada día, por lo cual la amplitud térmica diaria simulada a  
34 veces no es realista. Los diagnósticos previos realizados en base al generador de Verdin *et*  
35 *al.* (2016) demostraron que algunas propiedades de las series sintéticas producidas no re-  
36 flejaban fielmente características importantes para el análisis de las sequías, por ejemplo, la  
37 persistencia de secuencias de días secos y lluviosos. Por este motivo, en este trabajo solo se  
38 mantuvo la estructura general del modelo de Verdin *et al.* (2016) y se modificaron todos  
39 los algoritmos (modelos estadísticos) para obtener series sintéticas más consistentes con los  
40 registros históricos y para optimizar el proceso de cálculo.

## 41 2 Fundamento

42 A partir de los datos climáticos históricos, comienza el proceso de ajuste de un modelo  
43 estadístico que permita representar el comportamiento de cada una de las variables para  
44 cada localidad. El generador está dividido en cuatro modelos aditivos generalizados: dos para  
45 modelar la precipitación y dos para las temperaturas máxima y mínima, respectivamente.  
46 El concepto principal detrás de la generación de series sintéticas es que cada valor de una  
47 variable puede ser considerado como la suma de una componente climática (clima local) y  
48 otra componente meteorológica aleatoria o tiempo local (Kleiber et al., 2013), es decir

$$X_{i,s} = \text{clima local} + \text{tiempo local}$$

49 donde  $X_{i,s}$  corresponde al valor de la variable  $X$  en el día  $i$  en la localidad  $s$ ; clima local  
50 corresponde a un valor medio de la variable para el día  $i$  en la localidad  $s$  y tiempo local  
51 corresponde a un estado particular de la atmósfera en el día  $i$  en la localidad  $s$ . El componente  
52 climático es especificado a través del ajuste de cada uno de los cuatro modelos aditivos

53 del generador. El componente meteorológico corresponde a los residuos de los modelos (la  
 54 diferencia entre los valores históricos y el componente climático estimado), es decir, a la  
 55 variabilidad no explicada por los mismos. El proceso de ajuste culmina con los parámetros  
 56 ajustados para los cuatro modelos mencionados. Posteriormente, se generan datos para los  
 57 años a simular que corresponden a los valores medios de la distribución para cada día del año  
 58 (clima local). Luego, se simulan una serie de valores aleatorios, a partir de los residuos de  
 59 cada modelo, que corresponden a la variabilidad propia de cada realización (tiempo local).

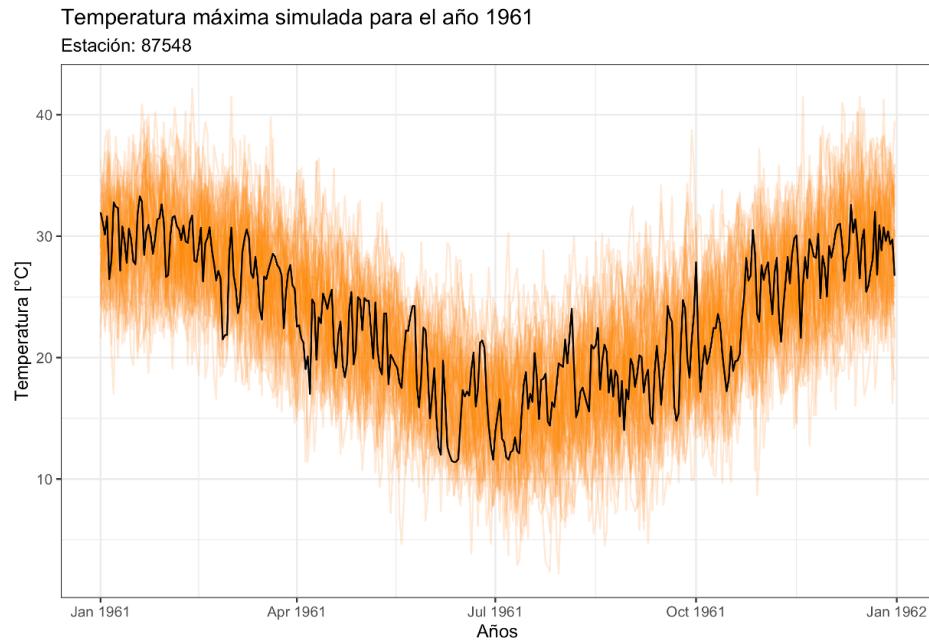


Figure 1: Componentes clima local y tiempo local

### 60 3 Tipos de series sintéticas

61 El generador GAMWGEN produce distintos tipos de series sintéticas. Desde el punto de  
 62 vista temporal, las series pueden ser **no condicionadas** (Figura 2.a), es decir, puramente  
 63 estacionarias; **pseudohistóricas** (Figura 2.b), que copian la variabilidad de baja frecuencia  
 64 y los cambios en la serie climática observada y **condicionadas** (Figura 2.c) que son series  
 65 forzadas a seguir la trayectoria de una salida de un modelo de cambio climático o una

66 trayectoria arbitraria definida por el usuario. Las siguientes Figuras muestran distintos  
67 ejemplos de lo mencionado.

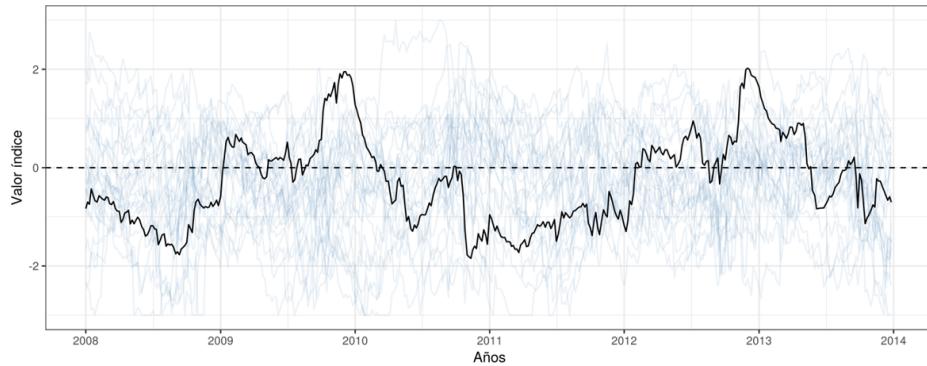


Figure 2: Tipos de series sintéticas: Series estacionarias

68 La línea negra corresponde a la serie temporal observada mientras que las distintas líneas  
69 celestes corresponden a realizaciones del generador. Al tratarse de series puramente estocás-  
70 ticas, son independientes entre sí.

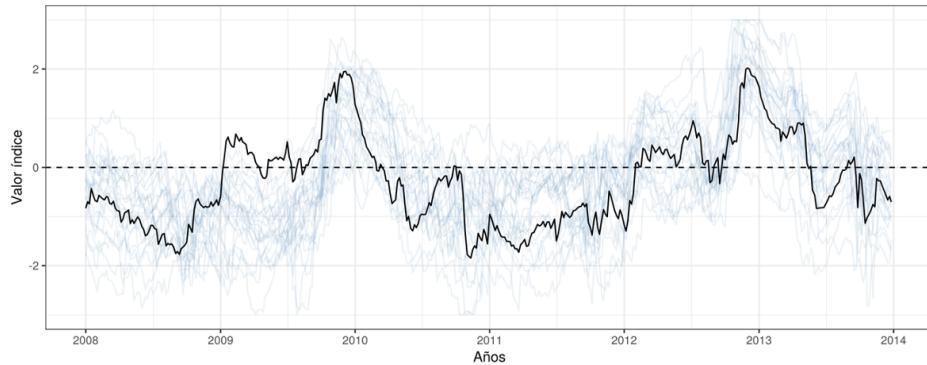


Figure 3: Tipos de series sintéticas: Series pseudohistóricas

71 La línea negra corresponde a la serie temporal observada mientras que las distintas líneas  
72 celestes corresponden a realizaciones del generador. En este caso, el generador fue forzado  
73 a seguir la trayectoria en los datos observados, por lo tanto, las series generadas siguen las  
74 variaciones de los datos observados.

75 En esta configuración el generador sigue una trayectoria arbitraria elegida por el usuario.  
76 En este caso es una trayectoria lineal por lo que la precipitación aumenta un determinado

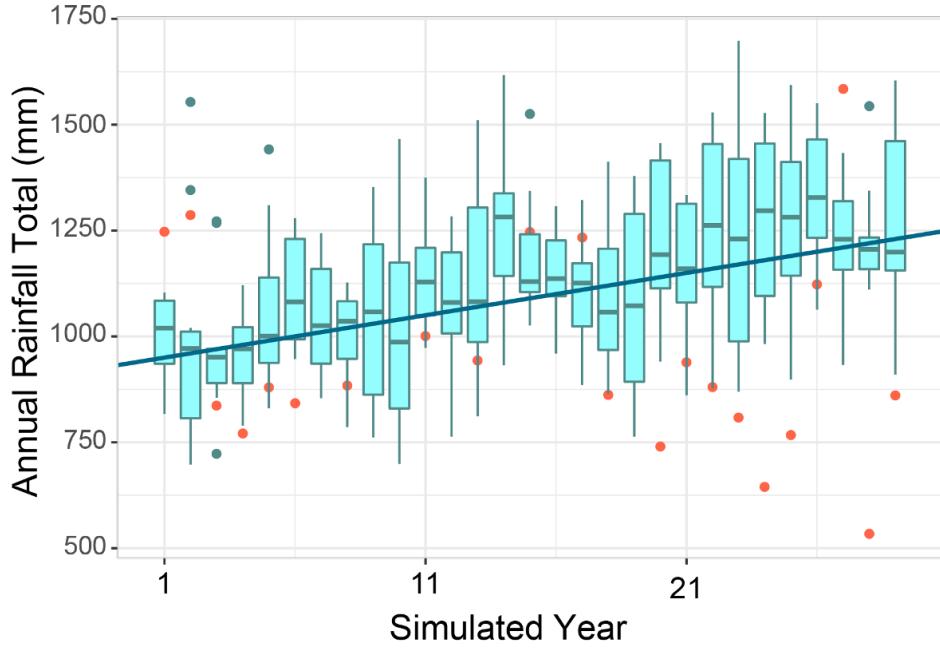


Figure 4: Tipos de series sintéticas: Series condicionadas

<sup>77</sup> porcentaje por año de manera lineal.

<sup>78</sup> Desde el punto de vista espacial se pueden generar series en estaciones meteorológicas, en  
<sup>79</sup> puntos que no se correspondan con estaciones o en una grilla regular. La Figura 3 muestra  
<sup>80</sup> un ejemplo de la generación en grillas sobre el Paraguay.

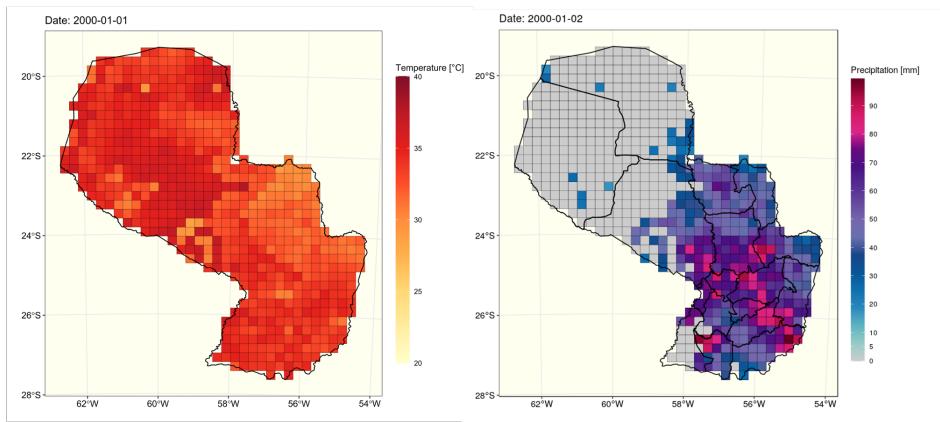


Figure 5: Tipos de series sintéticas: Datos grillados

<sup>81</sup> El panel de la izquierda muestra la temperatura máxima del día 1 de enero de 2000 para todo  
<sup>82</sup> el territorio del Paraguay mientras que el mapa del panel derecho muestra la precipitación

<sup>83</sup> diaria para el mismo día.

## <sup>84</sup> 4 Metodología

<sup>85</sup> Como su nombre indica, el generador está basado en Modelos Generalizados Aditivos (GAM,  
<sup>86</sup> por sus siglas en inglés). Como se mencionó en la Introducción, este generador está basado  
<sup>87</sup> en uno similar desarrollado por Verdin *et al.* (2016). Dicho generador estocástico utilizaba  
<sup>88</sup> modelos lineales generalizados (GLM, por sus siglas en inglés). Los GLM son modelos muy  
<sup>89</sup> interesantes ya que son fatalmente interpretables aunque carecen de la flexibilidad necesaria  
<sup>90</sup> para capturar complejos patrones como las variaciones estacionales. Por ello, en esta versión  
<sup>91</sup> se cambiaron los GLMs por GAMS. Estos nuevos modelos están siendo muy utilizados en  
<sup>92</sup> diversas áreas porque heredan lo mejor de los GLM pero son mucho más flexibles. En la  
<sup>93</sup> siguiente sección se describirán brevemente los GAMS.

### <sup>94</sup> 4.1 Introducción a los Modelos Aditivos Generalizados

<sup>95</sup> **4.1.0.1 Interpretabilidad vs Complejidad** Es posible crear modelos ajustados a los  
<sup>96</sup> datos que son lineales y relativamente fáciles de interpretar y explicar. Existe una línea  
<sup>97</sup> recta que representa la relación entre dos variables y atraviesa la nube de puntos. Es posible  
<sup>98</sup> graficar la relación y hacer predicciones a partir del modelo ajustado. Pero los modelos lineales  
<sup>99</sup> no siempre representan bien las relaciones entre variables ya que las relaciones no siempre  
<sup>100</sup> son lineales. Las predicciones no serían buenas a partir de estos modelos.

<sup>101</sup> En el otro extremo del espectro hay toda una serie de modelos de tipo “caja negra” como  
<sup>102</sup> las redes neuronales, random forest, árboles de regresión. Estos modelos son muy buenos  
<sup>103</sup> para predecir pero son muy difíciles de interpretar y de entender qué está sucediendo en el  
<sup>104</sup> sistema. Son muy útiles para clasificar pero no sirven para entender cómo una variable se  
<sup>105</sup> relaciona con el producto del modelo.

106 Los GAMs proveen un interesante punto intermedio ya que se pueden ajustar relaciones  
107 complejas y no lineales e interacciones pero estos modelos son explícitos y se pueden observar  
108 las relaciones entre variables y entender porque se produce un resultado determinado.

109 **4.1.0.2 Relaciones no lineales** En general, las relaciones entre variables de la natura-  
110 raleza no son lineales y adquieren patrones muy complejos. En la Figura 4 se muestra un  
111 ejemplo de datos sintéticas para ejemplificar este concepto.

```
library(mgcv)

set.seed(2) ## simulate some data...

dat <- gamSim(1, n=400, dist="normal", scale=0, verbose=FALSE)

dat <- dat[,c("y", "x0", "x1", "x2", "x3")]

ggplot2::ggplot(dat, ggplot2::aes(y=y, x=x2)) +
  ggplot2::geom_point() +
  ggplot2::theme_minimal()
```

112 Este diagrama de dispersión muestra dos variables que claramente se encuentran relacionadas  
113 pero no de manera lineal. Si se ajusta una regresión lineal simple a estos datos se obtienen  
114 los siguientes resultados.

```
ggplot2::ggplot(dat, ggplot2::aes(x2, y)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(method = 'lm', se = TRUE, ggplot2::aes(colour = "Lineal")) +
  ggplot2::scale_colour_manual(name = "", values = c("Steelblue")) +
  ggplot2::theme_minimal()
```

115 El modelo no es capaz de capturar las principales características de los datos.

116 Modelo lineal

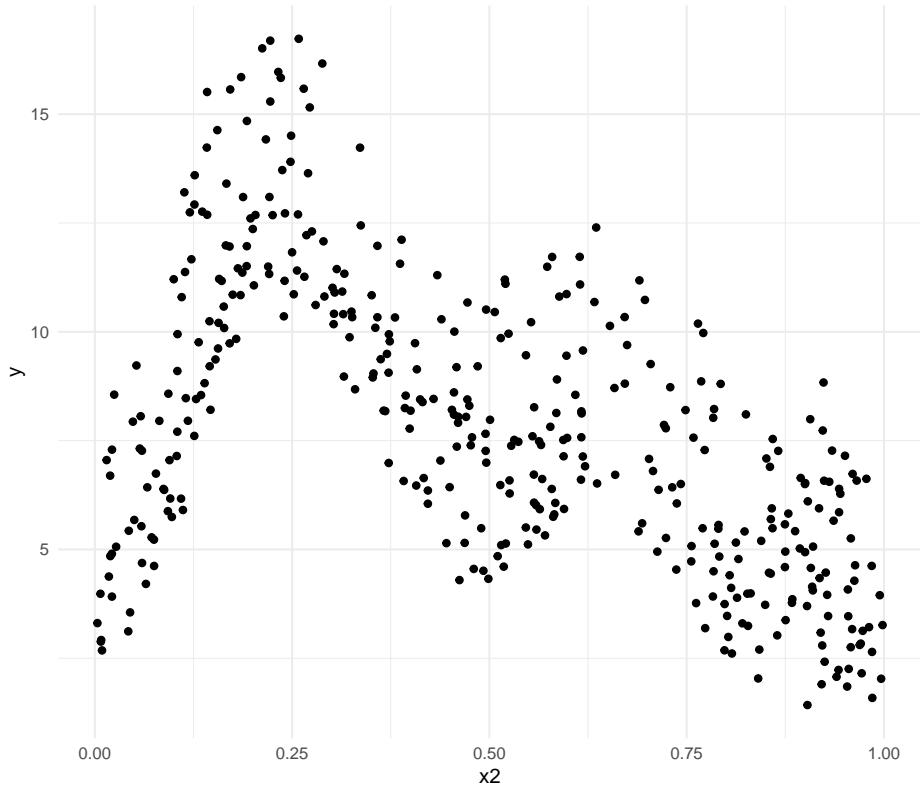


Figure 6: GAMs: Relaciones no lineales.

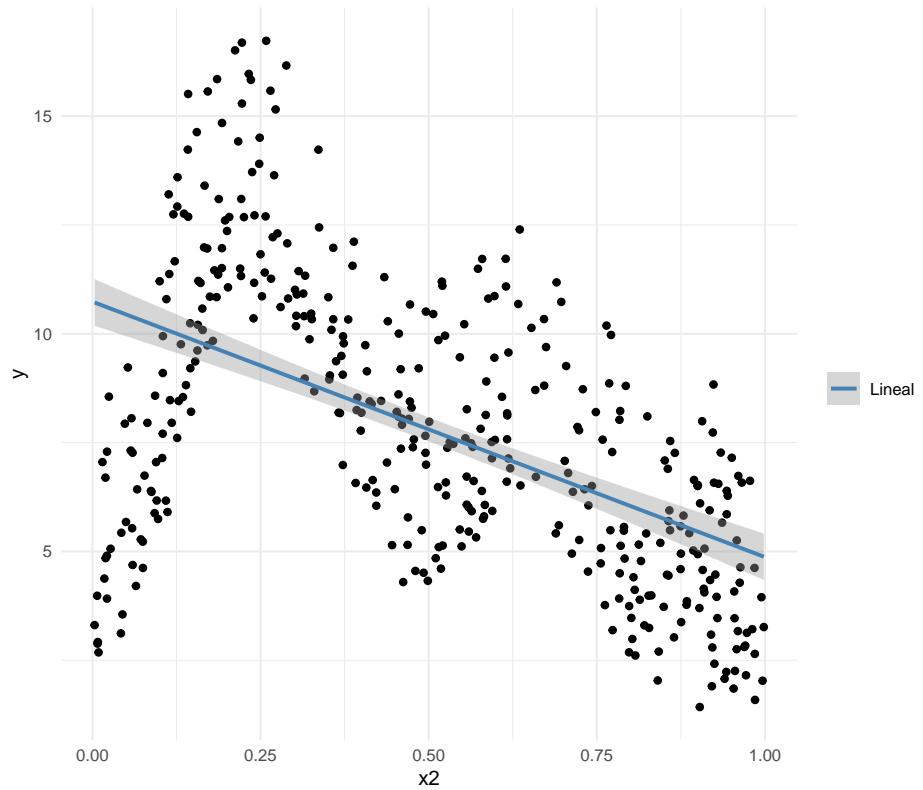


Figure 7: GAMs: Relaciones no lineales. Ajuste lineal. Tomado de (CITAR)

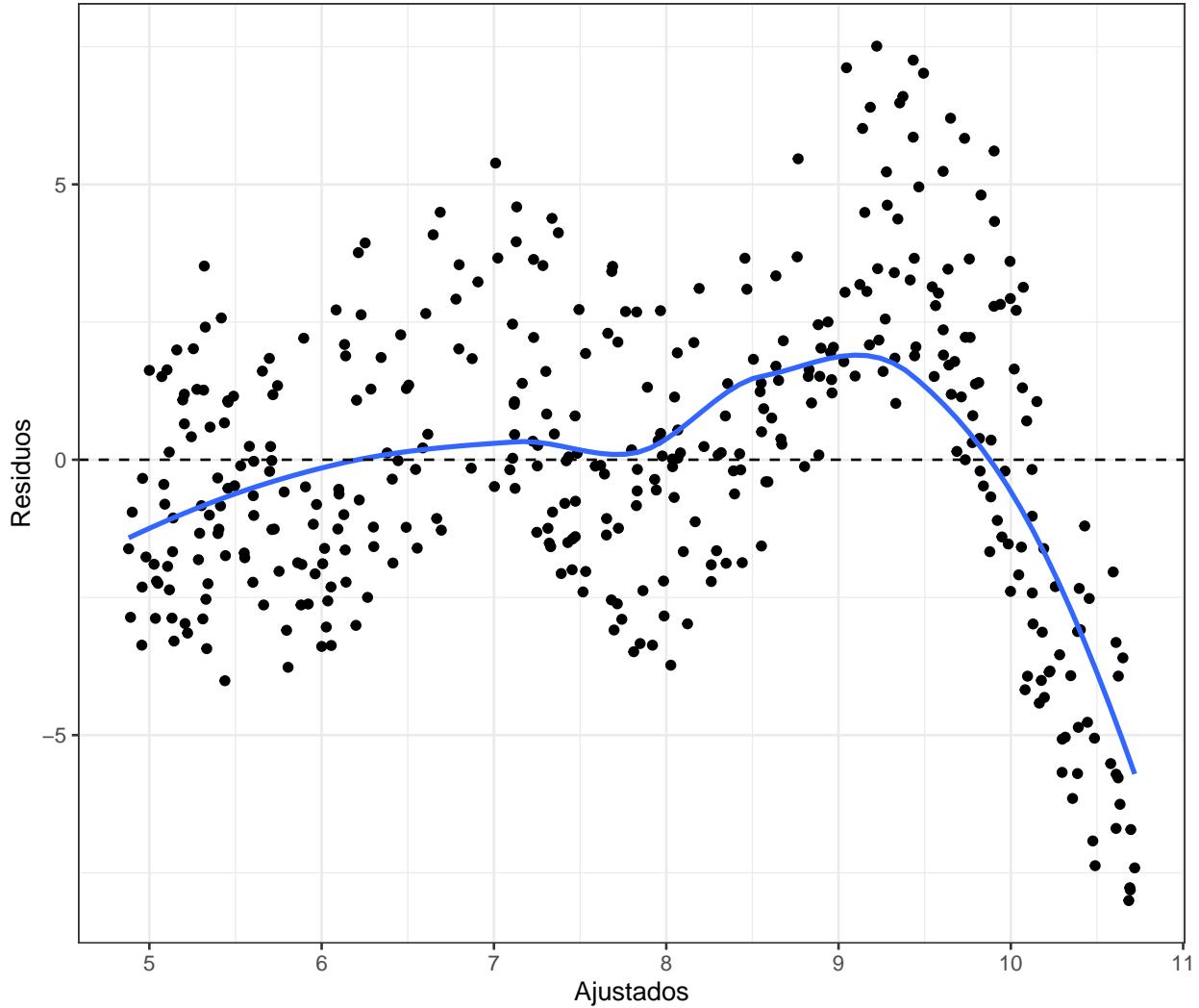
```
modelo_lineal <- lm(y ~ x2, data = dat)
```

117 Gráfico de residuos

```
residuals <- fortify(modelo_lineal)

ggplot2::ggplot(residuals, aes(x = .fitted, y = .resid)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(se = FALSE) +
  ggplot2::geom_hline(yintercept = 0, linetype = 'dashed') +
  ggplot2::theme_bw() +
  ggplot2::xlab('Ajustados') + ggplot2::ylab('Residuos')
```

118 ## `geom\_smooth()` using method = 'loess' and formula 'y ~ x'



119

120 Claramente los residuos tienen una tendencia y no están distribuidos aleatoriamente.

121 Sin embargo, al cambiar el modelo lineal por un `gam`.

122 Con un GAM se pueden ajustar los modelos a través de funciones suavizadas o splines que  
 123 pueden tomar casi cualquier forma. Al utilizar splines los GAMs pueden capturar diversos  
 124 tipos de relaciones no lineales y es por esto que son tan flexibles y adaptables a distintos  
 125 contextos.

```
modelo_gam <- gam(y ~ s(x2), data = dat)

residuals <- data.frame(fitted = modelo_gam$fitted.values,
                        resid = resid(modelo_gam)) %>%
```

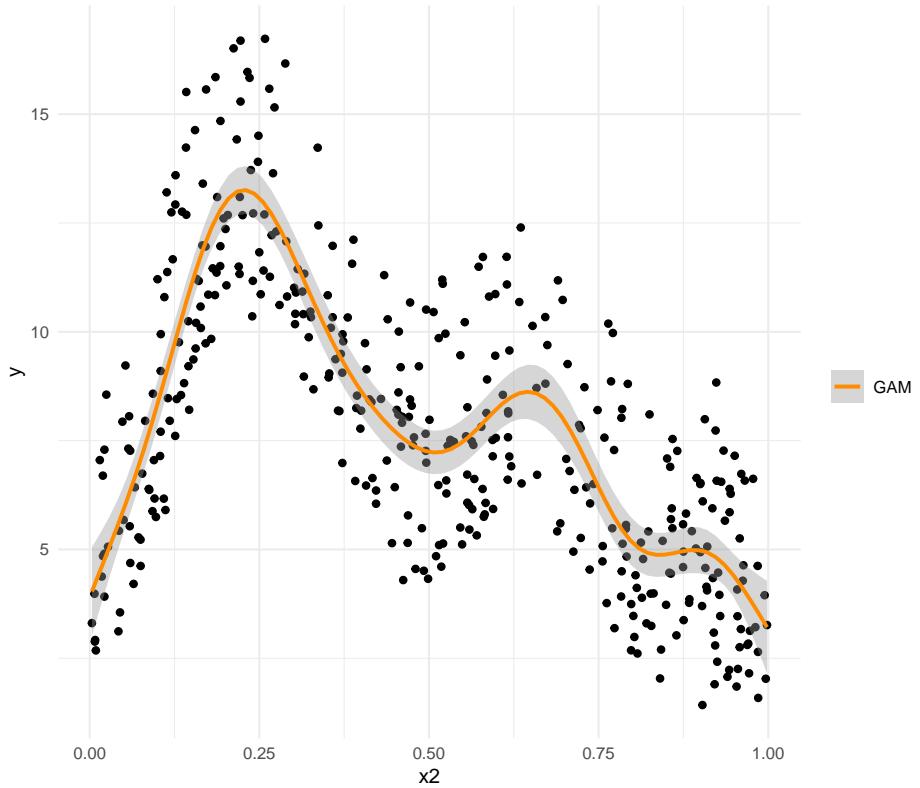


Figure 8: GAMs: Relaciones no lineales. Ajuste no lineal. Tomado de (CITAR)

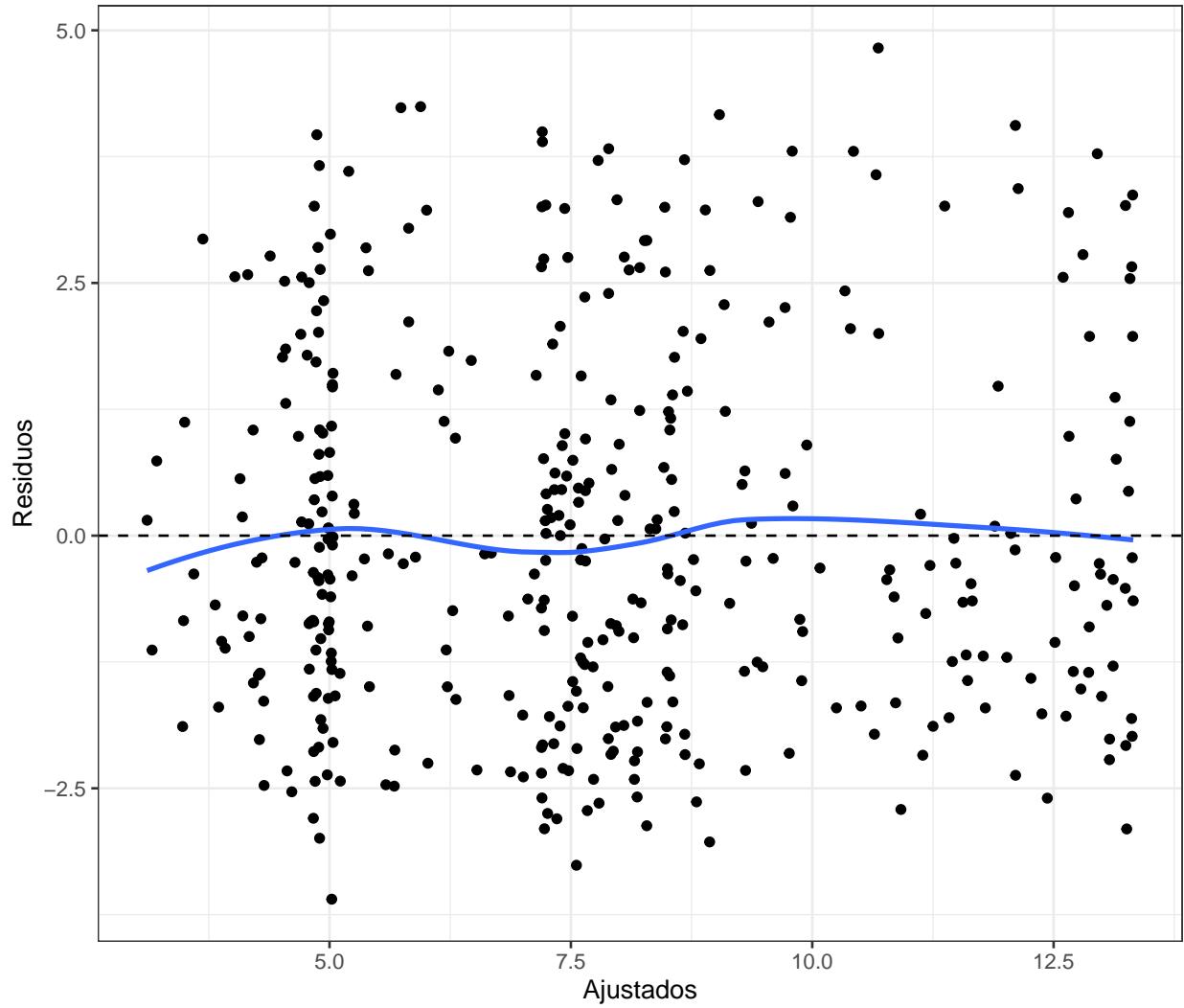
```

tibble::as_tibble()

ggplot2::ggplot(residuals, aes(x = fitted, y = resid)) +
  ggplot2::geom_point() +
  ggplot2::geom_smooth(se = FALSE) +
  ggplot2::geom_hline(yintercept = 0, linetype = 'dashed') +
  ggplot2::theme_bw() +
  ggplot2::xlab('Ajustados') + ggplot2::ylab('Residuos')

126 ## `geom_smooth()` using method = 'loess' and formula 'y ~ x'

```



127

128 Al utilizar un GAM los residuos tienen una distribución casi aleatoria sin una tendencia

<sup>129</sup> clara como si fue el caso del modelo lineal. Este es sólo en sencillo ejemplo del potencial que  
<sup>130</sup> tienen los GAMs para modelar complejas relaciones. Si se desea profundizar más en el tema  
<sup>131</sup> se recomienda revisar el libro de Simon Wood (2017)

## <sup>132</sup> 4.2 Modelación

<sup>133</sup> Como se mencionó anteriormente, el generador estocástico tiene como base cuatro modelos  
<sup>134</sup> generalizados, dos para temperaturas máxima y mínima y dos para la precipitación que  
<sup>135</sup> modelan la ocurrencia y los montos diarios.

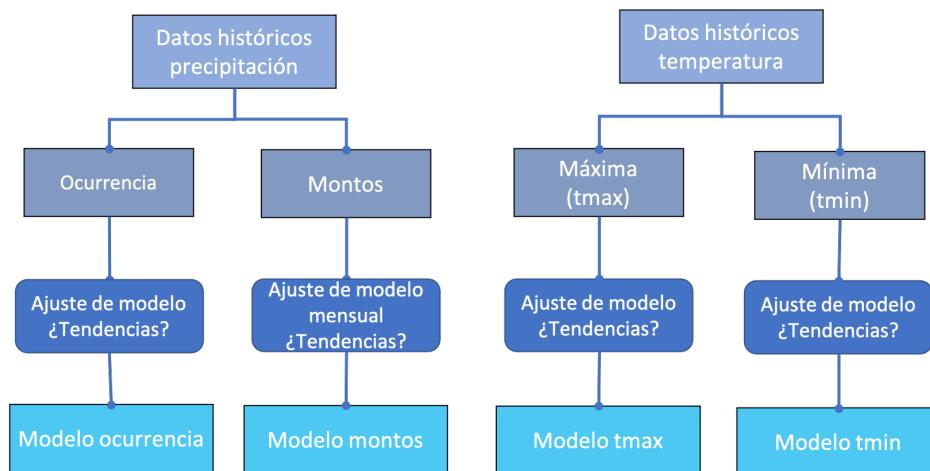


Figure 9: Modelos estadísticos

<sup>136</sup> Partiendo de la base de datos se extraen los datos diarios de temperaturas máxima y mí-  
<sup>137</sup> nima y precipitación. Los valores de temperaturas son utilizados para ajustar un GAM que  
<sup>138</sup> modelará la componente climática. Estos modelo puede incluir o no medias trimestrales que  
<sup>139</sup> servirán para condicionar el modelo a seguir un determinado comportamiento. Para el caso  
<sup>140</sup> de la precipitación el tratamiento es diferente. El fenómeno se divide en dos: ocurrencia de  
<sup>141</sup> precipitación y montos diarios. La ocurrencia se modela con GAM para toda la serie que  
<sup>142</sup> puede o no estar condicionado por totales trimestrales de lluvia. Los montos, en cambio, son  
<sup>143</sup> modelados a escala mensual. Es decir, se ajustan doce modelos, uno para cada mes del año.  
<sup>144</sup> Se utiliza esta modalidad para capturar mejor las características propias del ciclo estacional

<sup>145</sup> de la precipitación de cada región.

<sup>146</sup> A continuación se describirán cada uno de los modelos recién mencionados.

<sup>147</sup> **4.2.1 Clima local: Ocurrencia de lluvia**

<sup>148</sup> El ajuste del modelo de ocurrencia comienza con la definición de un día lluvioso. Según la  
<sup>149</sup> Organización Meteorológica Mundial (OMM), se considera como día lluvioso a aquel día con  
<sup>150</sup> una precipitación igual o mayor a 0.1 mm. Este valor, si bien es el sugerido por la OMM,  
<sup>151</sup> puede ser modificado por el usuario si así lo dispone. Una vez definida la ocurrencia de lluvia  
<sup>152</sup> se ajusta el modelo. La ocurrencia de lluvia es una variable de tipo binaria, es decir, un día  
<sup>153</sup> puede ser lluvioso (1) o seco (0), y es muy bien representada a través de una regresión probit  
<sup>154</sup> Kleiber *et al.* (2012). Este tipo de regresiones se basan en procesos latentes Gaussianos,  
<sup>155</sup>  $W_{s,t}$ , que se modelan con la siguiente relación:

$$O_{s,t} = \Pi_{\{W_{s,t} > 0\}}$$

<sup>156</sup> Si el proceso  $W_{s,t}$  es positivo, significa que lloverá en el día **t** y en la estación **s** y a ese día  
<sup>157</sup> se le asignará el valor 1. Si el proceso es negativo significa que no lloverá en el día **t** y en la  
<sup>158</sup> estación **s** y ese día se le asignará el valor 0. El uso de este tipo de procesos latentes está  
<sup>159</sup> justificado en que, en una región, la ocurrencia de lluvia en las distintas estaciones tenderá a  
<sup>160</sup> estar correlacionada. La función media del proceso latente Gaussiano es una regresión entre  
<sup>161</sup> variables que se expresa de la siguiente manera:

$$O_{s,t} = (s, O_{s,t-1}, f(doy(t)), f(ST(t)), f(lon, lat))$$

<sup>162</sup> donde  $O_{s,t}$  corresponde a la ocurrencia de lluvia en sitio y día determinado;  $s$  corresponde  
<sup>163</sup> al efecto del sitio **s** (ordenada al origen);  $O_{s,t-1}$  corresponde a la ocurrencia del día previo  
<sup>164</sup> que es un término autorregresivo;  $f(doy(t))$  corresponde a una función cíclica de los días

165 del año para considerar el efecto de la estacionalidad sobre la ocurrencia de lluvia;  $f(ST(t))$   
166 corresponde a una función suavizada de los acumulados estacionales de precipitación (solo  
167 se utiliza si se desea condicionar el modelo) y  $f(lon, lat)$  corresponde a las coordenadas del  
168 punto **s** (solo se utiliza en el modelo espacial). En la práctica, esta covariable se divide en  
169 cuatro, una para cada trimestre del año, asignándole el valor de 0 para los momentos fuera del  
170 respectivo trimestre. Es importante notar que para la ocurrencia de precipitación se usa un  
171 solo término autorregresivo. Este término es muy importante para la correcta modelización  
172 de las rachas secas y lluviosas mientras que el día del año incorpora la variabilidad intra-  
173 anual.

#### 174 4.2.2 Clima local: Montos diarios de lluvia

175 El modelo de montos diarios de lluvia difiere del anterior en que no se usan todos los datos  
176 de la serie, sino que se extraen de la base de datos los montos de precipitación únicamente  
177 para los días lluviosos. La intensidad de la precipitación para una localidad **s** y tiempo **t** es  
178 modelada como una variable aleatoria Gamma cuyos parámetros de forma y escala varían  
179 en el tiempo y en el espacio (Kleiber *et al.* (2012)). La función Gamma ha sido ampliamente  
180 utilizada en la región para la modelación de acumulados de lluvia. El modelo de montos  
181 puede ser expresado de la siguiente manera:

$$I_{s,t} = (s, O_{s,t-1}, f(ST(t)), f(lon, lat))$$

182 donde,  $I_{s,t}$  corresponde a los montos de precipitación en un sitio y día determinado;  $O_{s,t-1}$   
183 corresponde a la ocurrencia del día previo para considerar la autocorrelación temporal;  
184  $f(ST(t))$  corresponde a una función suavizada de los acumulados estacionales de precip-  
185 itación (solo se utiliza si se desea condicionar el modelo) y  $f(lon, lat)$  corresponde a las  
186 coordenadas del punto **s** (solo se utiliza en el modelo espacial). A diferencia del modelo  
187 anterior – que modela los años calendarios completos a través de la estacionalidad del día

188 del año – la serie de montos diarios es dividida en función del mes del año para ajustar una  
189 distribución a cada mes para obtener así parámetros mensuales más precisos. Gracias a esta  
190 modificación se obtienen parámetros que varían en el tiempo y en el espacio lo que permite  
191 capturar la variabilidad espacial de la precipitación. Al igual que en el modelo de ocurrencia,  
192 la inclusión del total trimestral es necesaria si se desean simular series condicionadas.

193 **4.2.3 Clima local: Temperatura**

194 Para el caso de la temperatura se utiliza una metodología similar a la utilizada para la precipi-  
195 tación, basada en Kleiber *et al.* (2013). A partir de los datos observados de temperatura se  
196 ajustan dos modelos: uno de máxima y otro de mínima. Ambos modelos utilizan las mismas  
197 variables para realizar el ajuste. El modelo puede ser expresado de la siguiente manera:

$$X_{s,t} = (s, O_{s,t}, O_{s,t-1}, f(T_{x_{(s,t-1)}}, T_{n_{(s,t-1)}}), f(doy(t)), f(SX(t), SN(t)), f(lon, lat))$$

198 donde,  $X_{s,t}$  corresponde a la temperatura máxima o mínima en un sitio y momento de-  
199 terminado;  $s$  corresponde al efecto del sitio  $s$  (ordenada al origen);  $O_{s,t}$  corresponde a la  
200 ocurrencia de lluvia en sitio y día determinado;  $O_{s,t-1}$  corresponde a la ocurrencia del día  
201 previo;  $f(T_{x_{(s,t-1)}}, T_{n_{(s,t-1)}})$  corresponde a una función suavizada de la interacción entre la  
202 temperatura máxima y mínima del día previo;  $f(doy(t))$  corresponde a una función cíclica  
203 de los días del año para considerar el efecto de la estacionalidad sobre la temperatura; El  
204 término  $f(SX(t), SN(t))$  corresponde a la interacción entre las medias estacionales de tem-  
205 peratura máxima y mínima y, como se explicó en la sección anterior, se incluyen para crear  
206 modelos condicionados y  $f(lon, lat)$  corresponde a las coordenadas del punto  $s$  (solo se uti-  
207 liza en el modelo espacial). La ocurrencia de lluvia es muy importante ya que en general  
208 los días lluviosos tienen temperaturas más bajas que los secos, sobre todo en verano, por lo  
209 que debe ser incluido en el ajuste. La Figura 6 es un ejemplo de esta influencia en Junín,  
210 en donde se muestra la amplitud térmica para cada mes del año en función del tipo de día,

211 seco o lluvioso.

#### 212 4.2.4 Tiempo local: Para una estación

213 **4.2.4.1 Precipitación** El **tiempo local** de la ocurrencia de precipitación se modela  
214 a través de los residuos de la regresión probit. Por definición estos residuos tienen una  
215 distribución  $X \sim \mathcal{N}(0, 1)$ . Por lo tanto, generar valores para el tiempo local es muy sencillo,  
216 solo se necesita de una función gaussiana que genere números aleatorios. La Figura muestra  
217 un ejemplo de la generación de clima local y tiempo local para una estación meteorológica  
218 de Argentina.

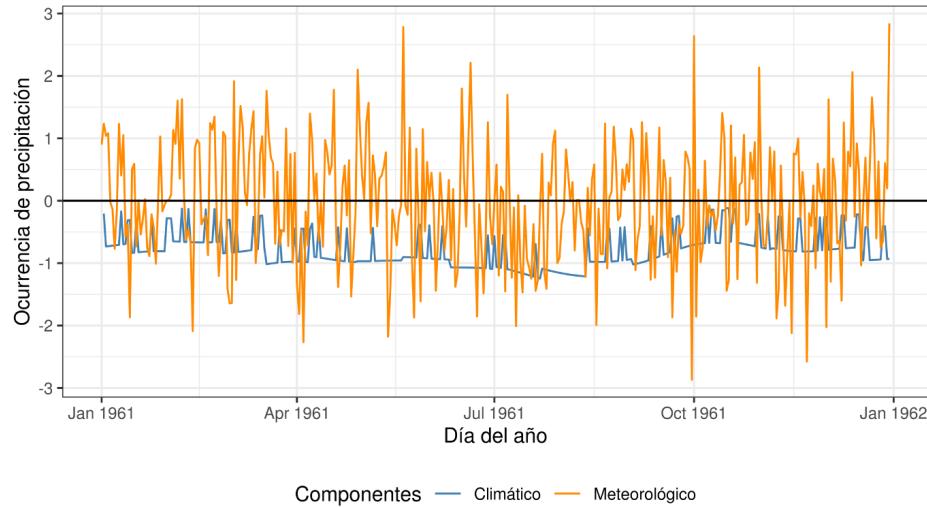


Figure 10: Tiempo local: precipitación

219 Esta Figura muestra sólo un año para mejorar la visualización pero su interpretación es válida  
220 para toda la longitud de la serie. La línea azul corresponde al clima local modelado a través  
221 del GAM y la naranja al ruido aleatorio creado a partir de una distribución  $X \sim \mathcal{N}(0, 1)$ .  
222 La sumatoria de ambos componentes determinarán si el día es lluvioso o no. Si la suma es  
223 positiva, lloverá, caso contrario será un día seco. Se puede observar en la línea azul un patrón  
224 estacional debido al régimen de precipitación tipo monzónico de esta región con picos de más  
225 días lluviosos durante el verano mientras que en invierno disminuyen marcadamente. Esta

226 misma serie temporal de números aleatorios serán la base para la generación de los montos  
227 de precipitación.

228 **4.2.4.2 Temperatura** El tiempo local de las temperaturas máxima y mínima se modela  
229 de una manera diferente. En este caso se toman los residuos de cada uno de los GAMs, es  
230 decir, la diferencia entre el valor ajustado por el modelo y el valor observado de temperatura.  
231 Además, como la temperatura está fuertemente influenciada por el tipo de día, días lluviosos  
232 tienden a tener una menor amplitud térmica que los días secos, los residuos se separan en  
233 función del tipo de día. Para capturar mejor el patrón estacional de la temperatura, los  
234 residuos se agrupan por mes y se ajusta un modelo bivariado que contemple los residuos de  
235 temperaturas máxima y mínima. El uso de un modelos bivariado es una alternativa muy  
236 interesante para mantener la consistencia entre ambas variables, es decir, que la temperatura  
237 máxima no supere a la mínima. La siguiente Figura es un ejemplo de las series de tiempo  
238 local para una localidad de Argentina.

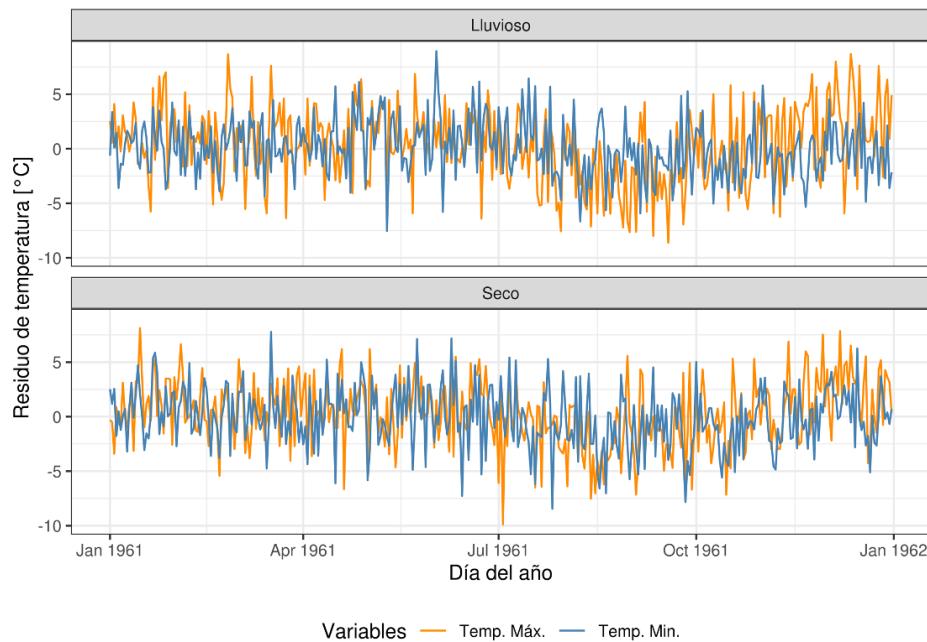


Figure 11: Tiempo local: temperatura

239 La figura esta dividida en dos paneles, el superior muestra el tiempo local para los días

<sup>240</sup> lluviosos y el inferior para los días secos. La línea naranja corresponde a los valores de  
<sup>241</sup> tiempo local para temperatura máxima y los azules para temperatura mínima. Si bien  
<sup>242</sup> ambas series difieren en magnitud y variabilidad, las dos tienden a variar conjuntamente.

#### <sup>243</sup> 4.2.5 Tiempo local: Para una grilla

<sup>244</sup> Para generar datos sobre una grilla regular o en puntos donde no hay datos para ajustar los  
<sup>245</sup> modelos se debe utilizar el modelo espacial. La generación del tiempo local en el espacio  
<sup>246</sup> es conceptualmente idéntico a la generación sobre estaciones meteorológicas sólo que utiliza  
<sup>247</sup> campos gaussianos para incluir la dimensión espacial. Los campos gaussianos se generan con  
<sup>248</sup> el paquete `RandomFields` y capturan la variabilidad espacial de cada una de las variables a  
<sup>249</sup> partir de su variograma.

<sup>250</sup> 4.2.5.1 Precipitación Para la precipitación se utiliza un modelo exponencial que utiliza  
<sup>251</sup> como parámetros el variograma ajustado a partir de los datos observados usando máxima  
<sup>252</sup> verosimilitud. Este modelo permite simular campos con una muy buena consistencia espacial.

<sup>253</sup> La ocurrencia de precipitación no ocurre de manera aislada en una región sino que, en general,  
<sup>254</sup> un evento lluvioso abarca una importante superficie. La siguiente Figura es un ejemplo de  
<sup>255</sup> los campos aleatorios generados sobre una grilla regular para una región de Argentina.

<sup>256</sup> La interpretación es análoga a la mostrada para una estación puntual. Los valores para cada  
<sup>257</sup> píxel se suman a la componente climática y así se determina si el día será lluvioso o no.  
<sup>258</sup> Este tipo de campos se generan para todos los días de la simulación y cada campo diario es  
<sup>259</sup> independiente del anterior.

<sup>260</sup> 4.2.5.2 Temperatura Al igual que para una estación, los campos gaussianos que se  
<sup>261</sup> generan son bivariados. Se utiliza el modelo Bivariado de Whittle Matern incluido en el  
<sup>262</sup> paquete `RandomFields`. La siguiente Figura es un ejemplo para un día de una realización  
<sup>263</sup> para grilla regular sobre Argentina.

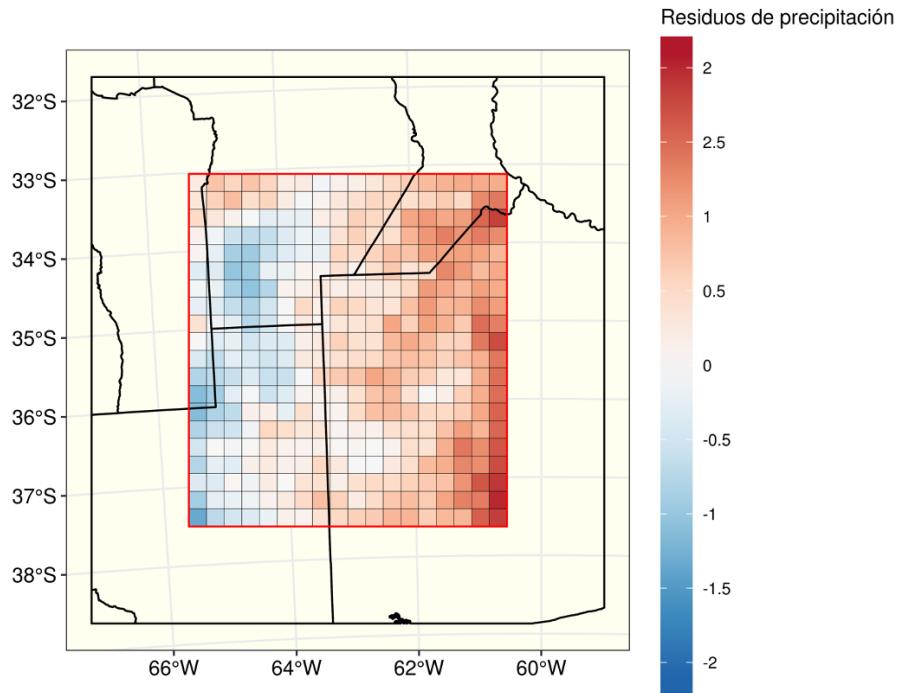


Figure 12: Tiempo local: precipitación sobre una grilla

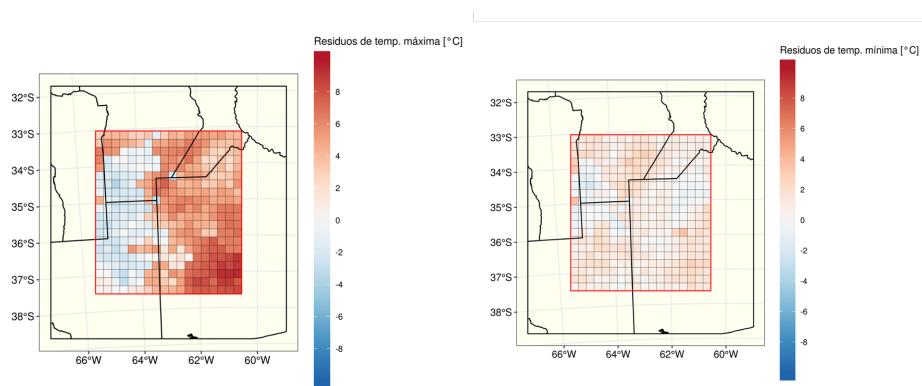


Figure 13: Tiempo local: temperatura sobre una grilla

264 Ambos campos se generan simultáneamente para temperaturas máxima y mínima y discrim-  
265 inando entre días secos y lluviosos. Esto quiere decir que en el proceso de creación de los  
266 campos se generan cuatro capas diferentes que luego se combinan en función del tipo de día  
267 de cada píxel resultando en dos campos integradores. Los valores para cada píxel se sumen  
268 a la componente climática y así se obtiene el valor final para cada día de la simulación.

## 269 5 Aplicación

270 En esta sección se mostrarán ejemplos de aplicación del generador para generar distintos  
271 tipos de series explicando las funciones necesarias y cada uno de los parámetros.

### 272 5.1 Instalar paquetes necesarios

273 El primer paso es comprobar que todos los paquetes necesarios estén instalados y si no es  
274 así, descargarlos e instalarlos.

### 275 5.2 Creación de directorios

276 El paquete tiene precargados algunos ejemplos de aplicación con datos reales de estaciones  
277 meteorológicas de la red del SISSA.

278 Para poder seguir este manual se deben crear directorios donde se guardarán los datos de  
279 entrada y salida.

- 280 • /input\_data: aquí se guardarán los datos meteorológicos y los metadatos de las esta-  
281 ciones
- 282 • /output\_data: aquí se guardarán los resultados de la simulación

283 Si estos directorios no existen, se crearán.

284 **5.3 Generación de series sintéticas para una sola estación meteo-**  
285 **rológica**

286 Este primer ejemplo consiste en la generación de series sintéticas para estaciones meteo-  
287 rológicas. Es decir, se generarán series para las mismas estaciones que fueron utilizadas en  
288 el ajuste de los distintos modelos. Por lo tanto, no se considera la componente espacial.  
289 El ejemplo está dividido en dos, en una primera parte se ajustará el modelo para generar  
290 series estacionarias y luego, en una segunda parte, se incluirán covariables estacionales para  
291 producir series pseudohistóricas. Para ambos ejemplos los datos son usados serán los mismos.

292 **5.3.1 Crear archivos de entrada**

293 El primer paso consiste en generar los set de datos de entrada que se descargaron al momento  
294 de instalar el paquete del generador estocástico. Estos datos son sólo a título demostrativo,  
295 si el usuario desea correr el modelo con sus propios datos deberá cambiar los objetos que se  
296 generarán en esta sección por los suyos y colocarlos en la carpeta `input_data`.

297 Los archivos necesarios son:

- 298     • `stations.csv`  
299     • `climate.csv`

300 Los datos meteorológicos se dividen en dos archivos separados: `stations.csv` y  
301 `climate.csv`. Los nombres de los mismos no deben ser necesariamente iguales a los  
302 usados aquí.

303 Los metadatos de las estaciones se alojan en el archivo `stations.csv`. Este archivo contiene  
304 la información de las estaciones meteorológicas que serán usadas en el ajuste del modelo.  
305 Las variables que deben ser incluidas en la tabla son:

- 306     • station\_id: número único para cada estación meteorológica. La variable debe ser  
 307       de tipo *integer*  
 308     • latitude: latitud en grados decimales. La variable debe ser de tipo *double*  
 309     • longitude: longitud en grados decimales. La variable debe ser de tipo *double*

310 La tabla puede tener más variables pero sólo se necesitan las anteriores.

311 A continuación se muestran la primera fila del dataset y los tipos de datos de cada una de  
 312 las variables.

*# Vista de los metadatos de la estación*

```
knitr::kable(stations[1,])
```

x	y	station_id	nombre	lat_dec	lon_dec	elev	pais_id
5001614	6256841	87448	Villa Reynolds Aero	-33.7181	-65.3737	486	AR

313  
 314 El objeto **stations** debe ser convertido de *tibble* a *sf*. El sistema de referencia espacial debe  
 315 ser planar. No es necesario un sistema de referencia espacial en particular, solamente las  
 316 coordenadas deben estar expresadas en metros.

*# Convertimos el objeto stations a sf y se convierte su proyección de WGS 1984 a*

*# POSGAR Argentina Faja 5.*

```
stations %<>%
  sf::st_as_sf(coords = c("lon_dec", "lat_dec"), crs = 4326) %>%
  sf::st_transform(crs = 22185)
```

317 La información climática se aloja en el archivo **climate.csv**. Este archivo contiene los datos  
 318 de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables que  
 319 deben ser incluidas en la tabla son:

- 320     • date: fecha del dato. La variable debe ser de tipo *date*

- 321     • `station_id`: número único para cada estación meteorológica. La variable debe ser  
 322       de tipo *integer*
- 323     • `prcp`: datos diarios de precipitación La variable debe ser de tipo *double*
- 324     • `tmax`: datos diarios de temperatura máxima. La variable debe ser de tipo *double*
- 325     • `tmin`: datos diarios de temperatura mínima. La variable debe ser de tipo *double*

326 A continuación se muestran las primeras cinco filas del dataset y los tipos de datos de cada  
 327 una de las variables.

```
knitr::kable(climate[1:10,])
```

date	station_id	tmax	tmin	prcp
1961-01-01	87448	37.4	13.5	0.6
1961-01-02	87448	27.4	14.3	23.9
1961-01-03	87448	26.6	13.5	0.0
1961-01-04	87448	31.0	11.7	6.0
328 1961-01-05	87448	27.0	14.1	0.0
1961-01-06	87448	26.3	11.3	0.0
1961-01-07	87448	34.1	12.0	6.7
1961-01-08	87448	32.8	15.9	0.0
1961-01-09	87448	37.6	16.1	0.0
1961-01-10	87448	26.9	4.6	0.0

329 Los nombres de las variables son importantes y deben ser siempre los mismos ya que el  
 330 modelo las reconocerá a partir de los mismos. Los nombres deben ser los siguientes:

- 331     • `date` : corresponde a la fecha del día en formato Date. El formato de la fecha para fa-  
 332       cilitar el reconocimiento por parte de R es “YYYY-MM-DD”, es decir, el año expresado  
 333       con cuatro dígitos y luego dos dígitos para el mes y dos para el día.

- 334 • `station_id`: Identificador único de cada una de las estaciones. Debe ser un número  
335 **entero**.
- 336 • `tmax`: temperatura máxima diaria expresada en °C.
- 337 • `tmin`: temperatura mínima diaria expresada en °C.
- 338 • `prcp`: precipitación diaria expresada en mm.

339 El orden de las variables no es importante pero, como se mencionó, si se deben respetar los  
340 nombres de cada una. En el caso de faltantes, no se utiliza ningún valor específico para los  
341 NAs, sólo se debe dejar ese valor vacío. Este archivo tiene un formato largo, es decir, las  
342 estaciones se deben colocar una debajo de la otra.

343 La generación de series sobre estaciones requiere de dos funciones básicas `local_fit` y  
344 `local_simulate`. Independientemente de si se incluyen totales trimestrales en el modelo,  
345 siempre se utilizan esas dos funciones.

### 346 5.3.2 Series sintéticas estacionarias

347 **5.3.2.1 Ajuste de los modelos** A continuación se mostrará como ajustar los cuatro  
348 modelos estadísticos para una sola estación meteorológica para generar series estacionarias  
349 donde cada realización es completamente independiente.

350 El ajuste del modelo local (en un punto) necesita de dos funciones: en una se define la  
351 configuración general del modelo y con la segunda se corre el modelo propiamente dicho.

352 Primero se crea un objeto con el control para el ajuste del simulador. Los argumentos son:

- 353 • `prcp_occurrence_threshold`: umbral de precipitación para un día lluvioso. La OMM  
354 recomienda un umbral de 0.1 mm para considerar un día como lluvioso.
- 355 • `avbl_cores`: cantidad de núcleos disponibles para la paralelización.
- 356 • `planar_crs_in_metric_coords`: sistema de coordenadas planar.

```

control_fit <- gamwgen::local_fit_control(
  prcp_occurrence_threshold = 0.1, # Umbral para la definición de días húmedos
  avbl_cores = 6, # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 22185) # Sistema de referencia espacial (en metros)

```

357 Luego se corre el ajuste para la estación meteorológica con la función `local_calibrate`.

358 Los argumentos de la función son:

- 359 • `climate`: datos meteorológicos observados para la estación
- 360 • `stations`: metadatos de las estaciones meteorológicas
- 361 • `seasonal_covariates`: datos agregados trimestrales. Si es NULL el ajuste será sin covariables y las series generadas serán estacionarias.
- 362 • `control`: objeto de control
- 363 • `verbose`: controla la impresión de mensajes en la consola. FALSE por defecto.

```

# Al correr la función se realiza el ajuste de los cuatro modelos
# para cada una de las estaciones. En este caso, por cuestiones de tiempo
# a cargar un objeto ya precalculado.

```

*# Si el usuario desea correrlo deberá ver la nota anterior.*

```

gamgen_fit <- gamwgen::local_calibrate(
  climate = climate, # Registro histórico de variables meteorológicas
  stations = stations, # Estaciones meteorológicas
  seasonal_covariates = NULL, # Totales trimestrales de precipitación
  control = control_fit, # Objeto de control
  verbose = FALSE) # Impresión de mensajes en la consola.

```

365 Para esta demostración, cargamos el objeto con el ajuste del modelo ya realizado.

```

# Copiamos el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/local/fit_local_unconditional.RData')) {

```

```

fs::file_copy(system.file('/autorun/local', "fit_local_unconditional.RData",
                         package = "gamwgen"),
              new_path = 'input_data/local/fit_local_unconditional.RData')

}

# Cargamos el archivo recientemente creado
load('input_data/local/fit_local_unconditional.RData')

# Clase del objeto con el ajuste del generador
class(gamgen_fit)

366 ## [1] "gamwgen"

# Contenido del modelo
names(gamgen_fit)

367 ## [1] "control"           "stations"          "climate"
368 ## [4] "crs_used_to_fit"    "start_climatology" "fitted_models"
369 ## [7] "models_data"        "models_residuals"   "statistics_threshold"
370 ## [10] "exec_times"

371 Dentro del objeto se guardan todo lo necesario para la simulación así como información
372 accesoria.

373     • control: copia de la configuración usada para calibrar el generador
374     • stations: estaciones meteorológicas utilizadas para la calibración
375     • climate: datos climáticos de cada uno de las estaciones
376     • seasonal_covariates: series temporales de totales trimestrales de precipitación y
377         medias trimestrales de temperaturas máxima y mínima.

```

- `crs_used_to_fit`: sistema de referencia espacial usado para proyectar
- `start_climatology`: climatología diaria de cada una de las variables de entrada.
- `fitted_models`: modelos ajustados, uno para cada variable: temperaturas máxima y mínima y ocurrencia y montos de precipitación.
- `models_data`: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- `models_residuals`: residuos de cada uno de los modelos. Es decir, la diferencia entre el valor ajustado por el modelo (clima local) y el valor observado en el día **i**
- `statistics_threshold`: umbrales de amplitud térmica diaria por mes. Si la amplitud simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de mantener la consistencia entre variables
- `exec_times`: tiempo de ejecución de cada una de las etapas del ajuste

389 Cada uno de los GAMs ajustados se almacenan en el objeto `gamgen_fit` y pueden ser  
 390 evaluados con la función `summary()`.

```
summary(gamgen_fit$fitted_models$`87448`$tmax_fit)
```

```
391 ##
392 ## Family: gaussian
393 ## Link function: identity
394 ##
395 ## Formula:
396 ## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +
397 ##       s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30)
398 ##
399 ## Parametric coefficients:
400 ##                               Estimate Std. Error t value Pr(>|t|)
401 ## (Intercept) 25.61343     0.03214   796.8    <2e-16 ***
```

```

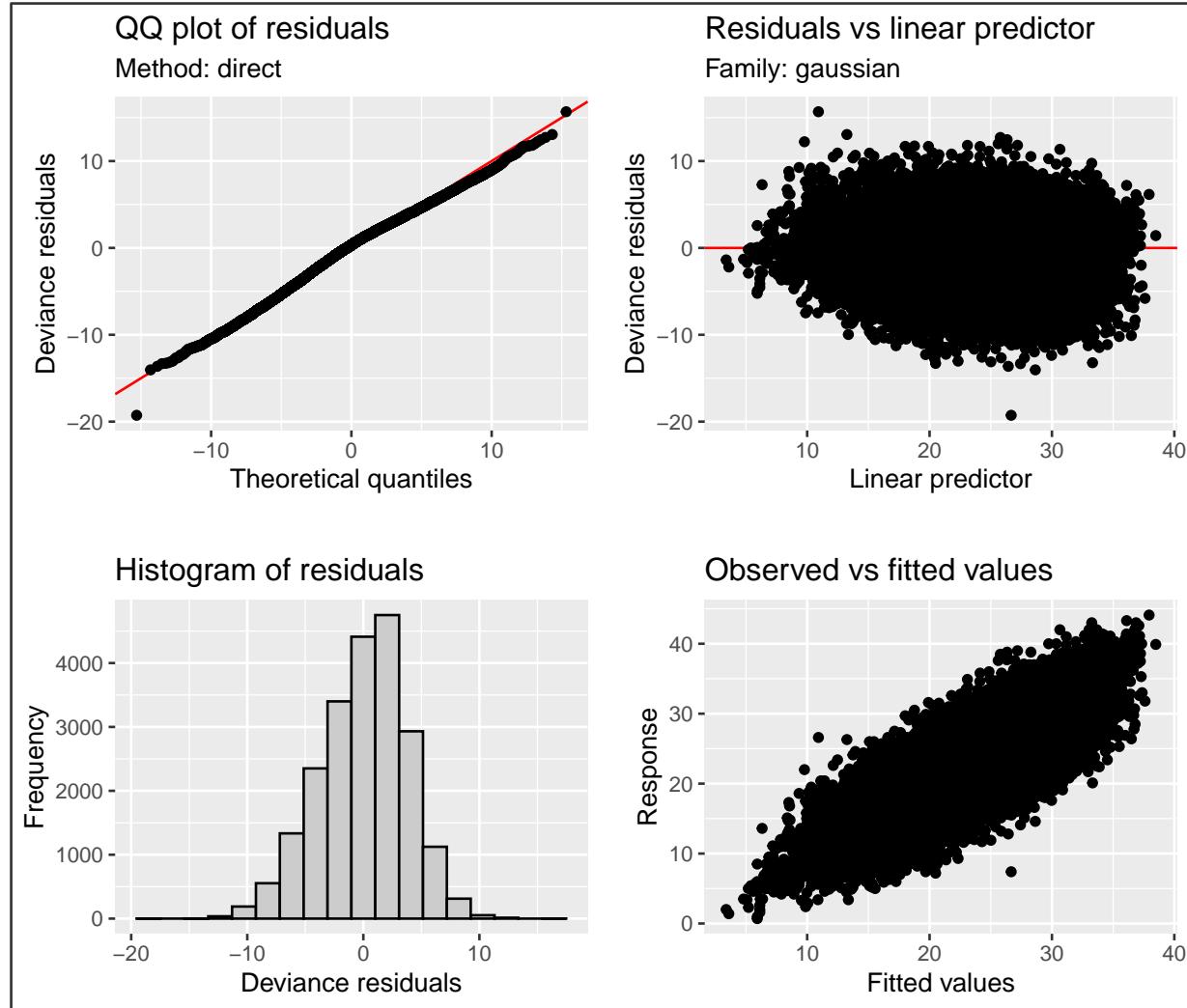
402 ## ---
403 ## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
404 ##
405 ## Approximate significance of smooth terms:
406 ##                               edf Ref.df      F p-value
407 ## s(tmax_prev,tmin_prev) 30.2469 38.71 247.1 <2e-16 ***
408 ## s(prcp_occ)           0.9989  1.00 1005.8 <2e-16 ***
409 ## s(prcp_occ_prev)     0.9995  1.00 2300.2 <2e-16 ***
410 ## s(doy)                12.8918 28.00 158.9 <2e-16 ***
411 ## ---
412 ## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
413 ##
414 ## R-sq.(adj) = 0.71 Deviance explained = 71.1%
415 ## fREML = 58955 Scale est. = 14.138 n = 21466

```

416 La función **summary** permite analizar los resultados del ajuste de cada uno de los modelos.  
417 Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la parte  
418 superior bajo el apartado **Formula** y la significancia de cada uno de los términos del modelo  
419 en la tabla inmediatamente inferior. Se puede observar que todos los términos son altamente  
420 significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la  
421 varianza explicada por el modelo y el valor de R-ajustado.  
422 El paquete **gratia** (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes  
423 de manera gráfica. Esta función toma los diagnósticos por defecto de la función **gam.check**  
424 del paquete **mgcv** (Wood, Simon (2001)) y los convierte en gráficos de la librería **ggplot2**  
425 (Wickham, Hadley(2011)) que son visualmente muy atractivos. La figura está dividida en  
426 cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo  
427 se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los

428 datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior  
429 derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los  
430 residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones  
431 en los residuos indicaría que el modelo no ha explicado algún componente importante de  
432 la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los  
433 residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El  
434 último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores  
435 ajustados y observados.

```
gratia::appraise(gamgen_fit$fitted_models`87448`$tmax_fit) +  
  ggplot2::theme_bw()
```



436

437 Estos diagnósticos exploratorios pueden aplicarse a cada uno de los modelos ajustados por la  
438 función `local_calibrate`.

439 Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar  
440 series sintéticas para la o las estaciones usadas para calibrar el generador.

441 **5.3.2.2 Generación de series** La generación de series sigue la misma estructura ante-  
442 rior, una función para configurar la generación y otra que realiza la generación propiamente  
443 dicha.

444 Los argumentos de la función de control son:

- 445     • **nsim**: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o  
 446         igual a 1.
- 447     • **seed**: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo  
 448         porque se guarda junto a los resultados.
- 449     • **avbl\_cores**: cantidad de núcleos disponibles para la paralelización.
- 450     • **use\_spatially\_correlated\_noise**: utilizar la generación estocástica espacialmente  
 451         correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron  
 452         más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible  
 453         calcular los variogramas necesarios para la generación de los campos aleatorios. Se  
 454         debe introducir un **boolean** (TRUE or FALSE).
- 455     • **use\_temporary\_files\_to\_save\_ram**: si se simulan muchas realizaciones o los recursos  
 456         informáticos son escasos, esta opción permite guardar los resultados de cada una de las  
 457         realizaciones en el disco liberando memoria RAM que quedará disponible para generar  
 458         nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno  
 459         único. Se debe introducir un **boolean** (TRUE or FALSE).
- 460     • **use\_temporary\_files\_to\_save\_ram**: esta opción permite eliminar los archivos tem-  
 461         porales creados para ahorrar RAM luego de terminar la generación de todas las simu-  
 462         laciones. Se debe introducir un **boolean** (TRUE or FALSE).

```
control_sim <- gamwgen::local_simulation_control(
  nsim = 10, # Cantidad de simulaciones a realizar
  seed = 1234, # Semilla para que los resultados sean reproducibles
  avbl_cores = 6, # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = FALSE,
  # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = FALSE,
  # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE)
```

```
# Borrar los resultados intermedios creados anteriormente
```

463 Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de  
464 simulación son:

- 465 • **model**: objeto con el resultado de la función `local_calibrate()`
- 466 • **simulation\_locations**: objeto tipo `sf` con la ubicación de las estaciones a simu-  
467 lar. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es  
468 necesario que todas estén presentes, se pueden generar series solo sobre algunas de  
469 ellas.
- 470 • **start\_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron  
471 covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria.  
472 Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de  
473 covariables, ni tampoco posterior. Se debe introducir una fecha en formato **date**
- 474 • **end\_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron co-  
475 variables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso  
476 contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables,  
477 tampoco puede ser posterior. Se debe introducir una fecha en formato **date**
- 478 • **control**: objeto de control creado con la función `control_sim()`.
- 479 • **output\_folder**: ruta al directorio donde se guardarán los resultados, tanto finales  
480 como intermedios.
- 481 • **output\_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad,  
482 el archivo generado es un archivo de texto en formato separado por comas (.csv)
- 483 • **seasonal\_covariates**: datos agregados trimestrales. Si el ajuste se realizó con co-  
484 variables, la generación también debe realizarse con ellas. Caso contrario se producirá  
485 un error. Se debe introducir un data frame con los valores agregados para las tres  
486 variables (precipitación y temperaturas máxima y mínima) pero no necesariamente  
487 deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de

algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar

- **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para
# cada una de las estaciones.

# En este caso, por cuestiones de tiempo, vamos a cargar un objeto
# con los resultados de la simulación

simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit, # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = NULL,
  # Covariables estacionales
  verbose = FALSE)

# Impresión de mensajes en la consola
```

- 492 Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y  
493 los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```
# Copiamos el archivo preajustado a nuestro directorio de trabajo

if (!fs::file_exists('output_data/simulated_local_unconditional.RData')) {

  fs::file_copy(system.file('/autorun/local', "simulated_local_unconditional.RData",
                           package = "gamwgen"),
                new_path = 'output_data/simulated_local_unconditional.RData')

}

# Cargamos el archivo recientemente creado

load('output_data/simulated_local_unconditional.RData')


# Clase del objeto con el ajuste del generador

class(simulated_climate)

494 ## [1] "list"           "gamwgen.climate"

# Contenido del modelo

names(simulated_climate)

495 ## [1] "nsim"
496 ## [2] "seed"
497 ## [3] "realizations_seeds"
498 ## [4] "simulation_points"
499 ## [5] "output_file_with_results"
500 ## [6] "output_file_fomart"
501 ## [7] "rdata_file_with_fitted_stations_and_climate"
502 ## [8] "exec_times"
```

503 La lista contiene los siguientes objetos:

- 504 • `nsim`: cantidad de realizaciones.
- 505 • `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la  
506 función de control.
- 507 • `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar  
508 los resultados.
- 509 • `simulation_points`: puntos donde se generaron las series sintéticas.
- 510 • `output_file_with_results`: nombre del archivo con los resultados.
- 511 • `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- 512 • `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos me-  
513 teorológicos observados que fueron utilizados en el ajuste. También se incluyen los  
514 metadatos de cada uno de esos puntos.
- 515 • `exec_times`: tiempo de ejecución de la generación.

516 Ahora veremos el formato del archivo de salida que contiene las series sintéticas.

517 Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones  
518 que forman parte de `local_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen  
519 los residuos y con la función `gamwgen:::generate_residuals_statistics` se calculan los  
520 parámetros.

521 En la tabla anterior se muestran los parámetros necesarios para generar el tiempo local de  
522 las temperaturas máxima y mínima.

```
gen_noise_params <- gamwgen:::generate_residuals_statistics(  
  models_residuals = gamgen_fit$models_residuals)
```

```
knitr::kable(gen_noise_params)
```

station_id	type_day	month	sd.tmax_residuals	sd.tmin_residuals	mean.tmax_residuals	mean.
87448	Wet	1	3.254469	2.543695	0.8526532	
87448	Dry	1	3.254469	2.543695	-0.3494070	
87448	Wet	2	3.352673	2.589353	0.7087247	
87448	Dry	2	3.352673	2.589353	-0.3295625	
87448	Dry	3	3.444108	2.678806	0.0577372	
87448	Wet	3	3.444108	2.678806	-0.2366143	
87448	Dry	4	3.526194	2.929321	0.1289181	
87448	Wet	4	3.526194	2.929321	-0.4627672	
87448	Wet	5	3.476241	3.248211	-0.9474860	
87448	Dry	5	3.476241	3.248211	0.2077445	
87448	Dry	6	3.716563	3.439488	0.1204929	
523	87448	Wet	6	3.716563	3.439488	-1.2864223
	87448	Dry	7	4.010964	3.536036	0.1423125
	87448	Dry	8	4.284256	3.412616	0.2143560
	87448	Wet	8	4.284256	3.412616	-2.0613069
	87448	Dry	9	4.296120	3.424206	0.2888032
	87448	Wet	9	4.296120	3.424206	-1.1867406
	87448	Dry	10	4.073819	3.168727	0.0124212
	87448	Wet	10	4.073819	3.168727	-0.1022864
	87448	Dry	11	3.836188	2.782351	-0.2913530
	87448	Wet	11	3.836188	2.782351	0.6726400
	87448	Dry	12	3.587819	2.697795	-0.4647068
	87448	Wet	12	3.587819	2.697795	0.9374758
	87448	Wet	7	4.010964	3.536036	-1.4985168

524

- station\_id: número único que identifica a cada estación meteorológica.

- 525 • type: tipo de día **lluvioso (Wet)** o **seco (Dry)**.
- 526 • month: número de mes para los que se calculan los parámetros
- 527 • sd.tmax\_residuals: desvío estándar de los residuos del modelo de temperatura máxi-
- 528 ma.
- 529 • sd.tmin\_residuals: desvío estándar de los residuos del modelo de temperatura míni-
- 530 ma.
- 531 • mean.tmax\_residuals: media de los residuos del modelo de temperatura máxima.
- 532 • mean.tmin\_residuals: media de los residuos del modelo de temperatura mínima \*
- 533 cov.residuals: covarianza de los residuos.
- 534 • var.tmax\_residuals: covarianza de los residuos del modelo de temperatura máxima.
- 535 • var.tmin\_residuals: covarianza de los residuos del modelo de temperatura mínima.
- 536 Losa parámetros para cada uno de los meses permiten generar valores de tiempo local para
- 537 las dos temperaturas a partir de una distribución normal multivariada.
- 538 A continuación se muestra un ejemplo para el año 2019 sólo para los días lluviosos.

```
fechas <- data.frame(
  date = seq(as.Date('2019-01-01'), as.Date('2019-12-31'), 'days')) %>%
  dplyr::mutate(dia = lubridate::day(date),
  mes = lubridate::month(date))

# Ejemplo de generación de ruido para el mes de enero

tmax_dry <- purrr::map2_dfr(
  .x = fechas$dia,
  .y = fechas$mes,
  .f = function(dia, mes) {
    result_tmax_dry <- control_sim$temperature_noise_generating_function(
```

```

simulation_points = stations %>%
  dplyr::filter(., station_id == '87448'),
  gen_noise_params = gen_noise_params,
  month_number = mes,
  selector = 'tmax_dry',
  seed = NULL)

result_tmax_dry <- result_tmax_dry %>%
  dplyr::mutate(dia = dia,
  mes = mes)

}

) %>%
sf::st_set_geometry(NULL) %>%
dplyr::mutate(date = as.Date(paste0('2019-', mes, '-', dia))) %>%
dplyr::select(-mes, -dia)

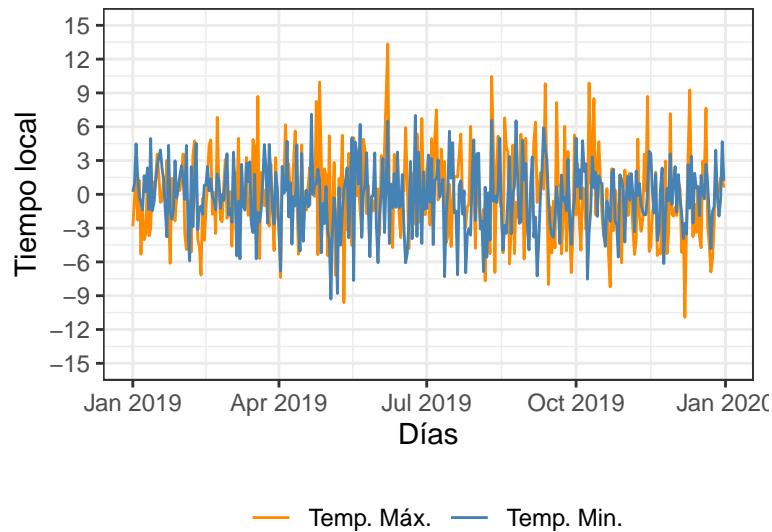
ggplot2::ggplot(data = tmax_dry %>%
tidyr::gather(residuo, valor, -date),
  ggplot2::aes(x = date, y = valor, color = residuo)) +
  ggplot2::scale_y_continuous(limits = c(-15, 15),
  breaks = seq(-15, 15, 3),
  name = 'Tiempo local') +
  ggplot2::scale_x_date(name = 'Días') +
  ggplot2::scale_color_manual(values=c("DarkOrange", "Steelblue"),
  labels = c("Temp. Máx.", "Temp. Min.)) +
  ggplot2::geom_line() +

```

```

ggplot2::theme_bw() +
ggplot2::theme(legend.position="bottom",
               legend.title = ggplot2::element_blank())

```



539

540 La línea naranja corresponde a la serie para temperaturas máximas y la azul para temper-  
541 aturas mínimas.

```

# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(
  here::here('output_data/local/simulated_local_unconditional.csv'))

# Primeras filas del objeto de salidas
knitr::kable(simulated_climate[1:10,])

```

realization	station_id	point_id	longitude	latitude	date	tmax	tmin	prcp	
1	87448	1	5001636	6256577	2019-01-01	27.19266	5.464899	0	
1	87448	1	5001636	6256577	2019-01-02	30.57296	10.266542	0	
1	87448	1	5001636	6256577	2019-01-03	33.36446	15.180860	0	
1	87448	1	5001636	6256577	2019-01-04	34.05627	15.473379	0	
542	1	87448	1	5001636	6256577	2019-01-05	32.69240	15.270145	0
1	87448	1	5001636	6256577	2019-01-06	30.56683	14.797530	0	
1	87448	1	5001636	6256577	2019-01-07	29.32913	12.624675	0	
1	87448	1	5001636	6256577	2019-01-08	34.09988	13.816714	0	
1	87448	1	5001636	6256577	2019-01-09	35.03660	19.596069	0	
1	87448	1	5001636	6256577	2019-01-10	37.52834	19.323512	0	

543 El resultado de la generación es un archivo .csv que contiene la siguiente información:

- 544 • **realization**: número de realización. Es un valor entero entre 1 y la cantidad de
- 545 realizaciones definida por el usuario.
- 546 • **station\_id**: número único de identificación de la estación meteorológica o del punto
- 547 arbitrario.
- 548 • **date**: fechas de cada uno de los días de la simulación.
- 549 • **tmax**: valores de temperatura máxima generada expresada en °C.
- 550 • **tmin**: valores de temperatura mínima generada expresada en °C.
- 551 • **prcp**: valores de precipitación diaria generada expresada en mm.

552 La siguiente Figura muestra un ejemplo de las series de temperaturas máximas y mínimas  
553 generadas.

```
# Grafico de temperatura máxima
tmax_plot <- ggplot2::ggplot() +
  ggplot2::geom_line(data = simulated_climate,
```

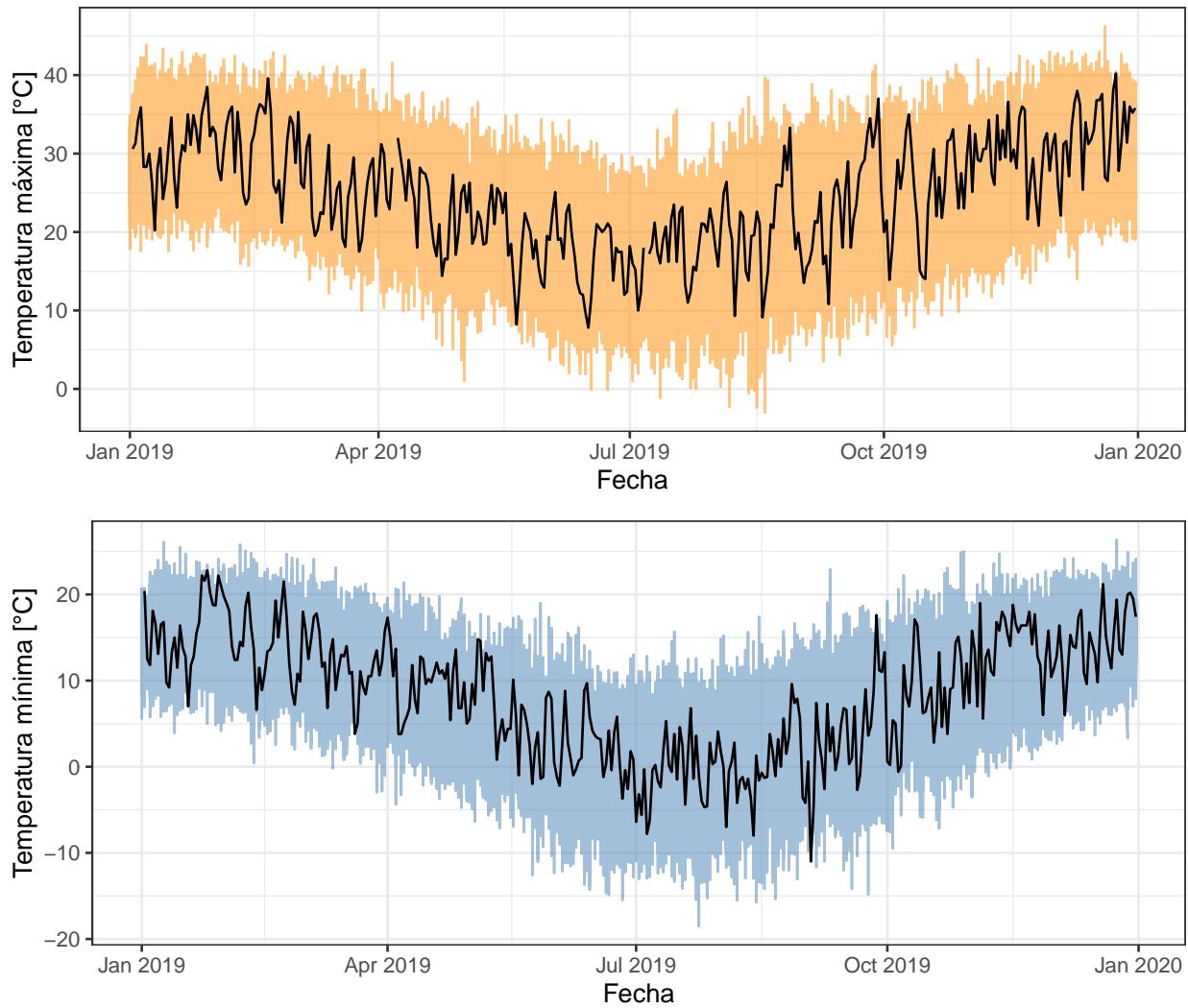
```

ggplot2::aes(x = date, y = tmax, color = realization),
alpha = 0.5, color = 'DarkOrange') +
ggplot2::geom_line(data = climate %>%
  dplyr::filter(date > as.Date('2019-01-01')),
  ggplot2::aes(x = date, y = tmax)) +
ggplot2::labs(x = 'Fecha', y = 'Temperatura máxima [°C]') +
ggplot2::theme_bw() +
ggplot2::theme(legend.position = "none")

# Grafico de temperatura mínima
tmin_plot <- ggplot2::ggplot() +
  ggplot2::geom_line(data = simulated_climate,
  ggplot2::aes(x = date, y = tmin, color = realization),
  alpha = 0.5, color = 'Steelblue') +
  ggplot2::geom_line(data = climate %>%
    dplyr::filter(date > as.Date('2019-01-01')),
    ggplot2::aes(x = date, y = tmin)) +
  ggplot2::labs(x = 'Fecha', y = 'Temperatura mínima [°C]') +
  ggplot2::theme_bw() +
  ggplot2::theme(legend.position = "none")

cowplot::ggdraw() +
  cowplot::draw_plot(tmax_plot, x = 0, y = 0.5, width = 1, height = .5) +
  cowplot::draw_plot(tmin_plot, x = 0, y = 0, width = 1, height = .5)

```



554

555 En el panel superior se muestra la temperatura máxima observada (negra) y las temperaturas  
 556 sintéticas (naranja) para el año 2019. En el inferior se muestra la temperatura mínima diaria  
 557 observada (negra) y cada una de las realizaciones (azul).

558 **5.3.3 Series sintéticas pseudohistóricas**

559 **5.3.3.1 Ajuste de los modelos** A continuación se mostrará como ajustar los cuatro  
 560 modelos estadísticos para una sola estación meteorológica para generar series **pseudo-**  
 561 **históricas** donde cada realización copia las variaciones de baja frecuencia e la serie ob-  
 562 servada. El ajuste del modelo local (en un punto) necesita de dos funciones: en una se define

563 la configuración general del modelo y con la segunda se corre el modelo propiamente dicho.

564 Primero se crea un objeto con el control para el ajuste del simulador. Los argumentos son:

- 565 • `prcp_occurrence_threshold`: umbral de precipitación para un día lluvioso. La OMM  
566 recomienda un umbral de 0.1 mm para considerar un día como lluvioso.
- 567 • `avbl_cores`: cantidad de núcleos disponibles para la paralelización.
- 568 • `planar_crs_in_metric_coords`: sistema de coordenadas planar.

```
control_fit <- gamwgen::local_fit_control(  
  prcp_occurrence_threshold = 0.1, # Umbral para la definición de días húmedos  
  avbl_cores = 6, # Cantidad de núcleos disponibles  
  planar_crs_in_metric_coords = 22185) # Sistema de referencia espacial (en metros)
```

569 Al tratarse de un modelo que ajusta condicionado por la variabilidad de baja frecuencia  
570 preexistente en los datos observados es necesario la agregación de las variables diarias en to-  
571 tales trimestrales de precipitación y medias trimestrales de temperaturas máxima y mínima.

572 Esta operación puede realizarse con la función `summarise_seasonal_climate` incluida en  
573 el paquete. Esta función, además de agregar los datos, permite la imputación de faltantes.

574 Se toleran una cierta cantidad que puede ser determinada por el usuario. El método de  
575 imputación utilizado es el `imputePCA()` de la librería `missMDA`.

```
# Agregación de valores diarios  
  
seasonal_covariates <- gamwgen::summarise_seasonal_climate(climate, umbral_faltantes = 0)  
  
# Se muestran las primeras cinco filas  
knitr::kable(seasonal_covariates[1:10,])
```

station_id	year	season	seasonal_prcp	seasonal_tmax	seasonal_tmin	
87448	1961	1	273.7	30.98764	14.5134831	
87448	1961	2	236.0	24.47473	8.4670330	
87448	1961	3	11.3	19.04565	1.3456522	
87448	1961	4	234.5	24.66235	7.7611765	
576	87448	1962	1	402.4	29.92333	14.2455556
87448	1962	2	187.2	24.30440	7.9868132	
87448	1962	3	50.9	17.21957	-0.8978261	
87448	1962	4	179.7	25.39560	7.8472527	
87448	1963	1	306.8	29.09101	14.1404494	
87448	1963	2	82.0	25.46196	8.5695652	

577 Cabe mencionar que con esta función las valores se agregan por trimestre considerando la  
 578 siguiente definición:

- 579 • Verano: Diciembre, Enero y Febrero
- 580 • Otoño: Marzo, Abril y Mayo
- 581 • Invierno: Junio, Julio y Agosto
- 582 • Primavera: Septiembre, Octubre y Noviembre

583 Los valores también se podrían agregar siguiendo otra definición de estaciones pero en ese  
 584 caso, el usuario debería hacerlo por su cuenta. Algunas funciones útiles para hacerlo son  
 585 las disponibles en el paquete `lubridate` como `quarter()` que permite definir el mes de  
 586 comienzo de los trimestres. Para estas variables los nombres también son importantes por  
 587 lo que deben respetarse los mostrados anteriormente.

588 La siguiente Figura muestra los totales trimestrales de precipitación para la estación mete-  
 589 orológica del ejemplo.

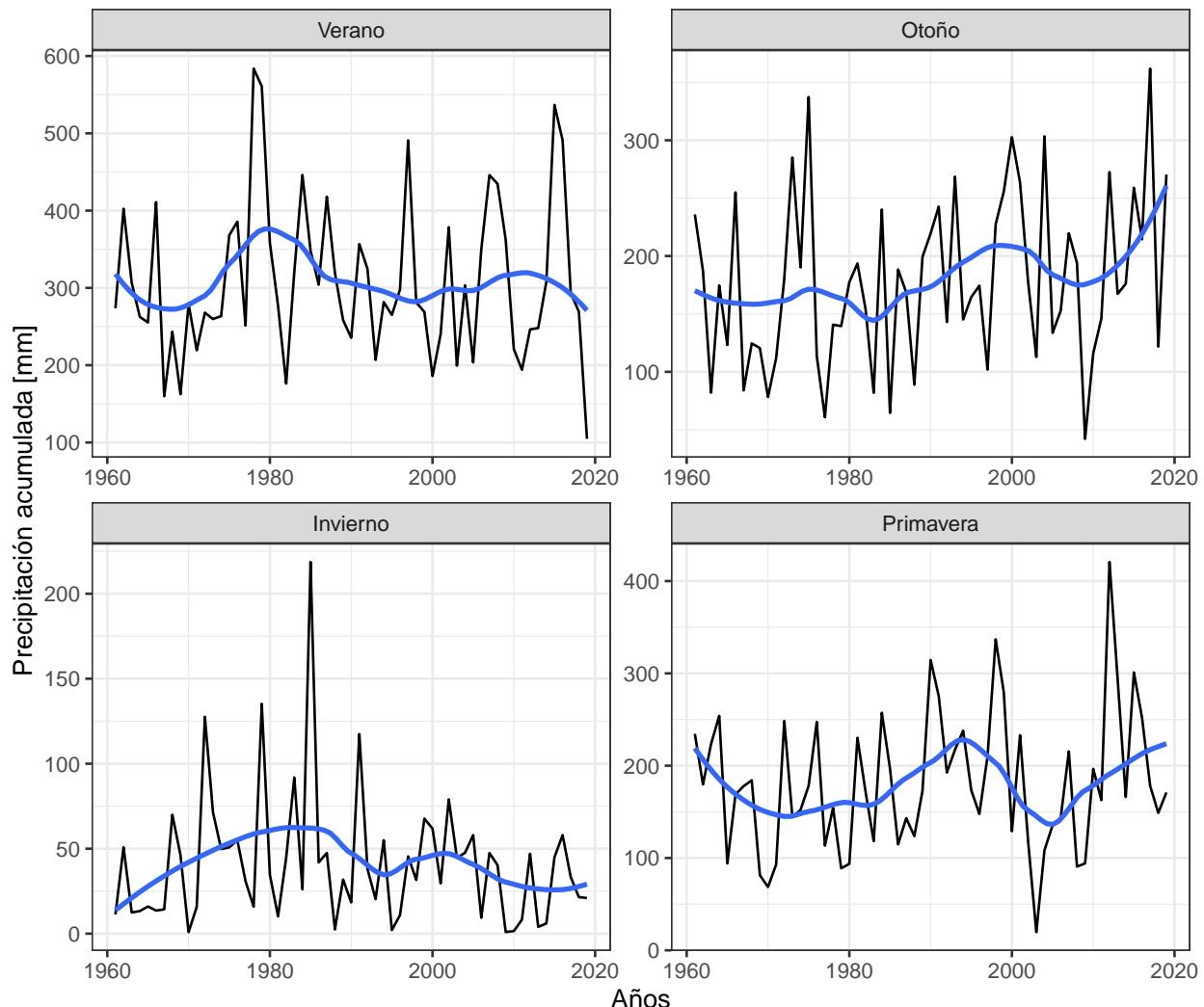
```
ggplot2::ggplot(data = seasonal_covariates %>%
```

```

dplyr::mutate(season = factor(season,
                               labels = c('Verano', 'Otoño', 'Invier
ggplot2::aes(x = year, y = seasonal_prcp, group = 1)) +
  ggplot2::geom_line() +
  ggplot2::geom_smooth(se = FALSE, span = 0.5) +
  ggplot2::facet_wrap(~season, scales = 'free') +
  ggplot2::labs(x = 'Años', y = 'Precipitación acumulada [mm]') +
  ggplot2::theme_bw()

590 ## `geom_smooth()` using method = 'loess' and formula 'y ~ x'

```



591

592 Cada uno de los paneles muestra la precipitación acumulada por trimestre junto a una  
593 regresión local (loess) en azul. Al incluir estos totales en el modelo, las series generadas  
594 tenderán a seguir la variabilidad observada, es decir, años secos generarán realización con  
595 valores inferiores a la media y viceversa.

596 Luego se corre el ajuste para la estación meteorológica con la función `local_calibrate`.

597 Los argumentos de la función son:

- 598 • `climate`: datos meteorológicos observados para la estación
- 599 • `stations`: metadatos de las estaciones meteorológicas
- 600 • `seasonal_covariates`: datos agregados trimestrales. Si es NULL el ajuste será sin  
601 covariables y las series generadas serán estacionarias.
- 602 • `control`: objeto de control
- 603 • `verbose`: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza el ajuste de los cuatro modelos para
# cada una de las estaciones. En este caso, por cuestiones de tiempo
# a cargar un objeto ya precalculado.
```

```
# Si el usuario desea correrlo deberá ver la nota anterior.
```

```
gamgen_fit <- gamwgen::local_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
```

```
verbose = FALSE)
```

```
# Impresión de mensajes en la consola.
```

- 604 Para esta demostración, cargamos el objeto con el ajuste del modelo ya realizado.

```
# Copiamos el archivo preajustado a nuestro directorio de trabajo
```

```
if (!fs::file_exists('input_data/local/fit_local_conditional.RData')) {  
  fs::file_copy(system.file('/autorun/local', "fit_local_conditional.RData",  
    package = "gamwgen"),  
    new_path = 'input_data/local/fit_local_conditional.RData')  
}
```

```
# Cargamos el archivo recientemente creado
```

```
load('input_data/local/fit_local_conditional.RData')
```

```
# Clase del objeto con el ajuste del generador
```

```
class(gamgen_fit)
```

- 605 ## [1] "gamwgen"

```
# Contenido del modelo
```

```
names(gamgen_fit)
```

```
606 ## [1] "control"           "stations"          "climate"  
607 ## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"  
608 ## [7] "fitted_models"       "models_data"       "models_residuals"  
609 ## [10] "statistics_threshold" "exec_times"
```

- 610 Dentro del objeto se guardan todo lo necesario para la simulación así como información

611 accesoria.

```

612     • control: copia de la configuración usada para calibrar el generador
613     • stations: estaciones meteorológicas utilizadas para la calibración
614     • climate: datos climáticos de cada uno de las estaciones
615     • seasonal_covariates: series temporales de totales trimestrales de precipitación y
616         medias trimestrales de temperaturas máxima y mínima.
617     • crs_used_to_fit: sistema de referencia espacial usado para proyectar
618     • start_climatology: climatología diaria de cada una de las variables de entrada.
619     • fitted_models: modelos ajustados, uno para cada variable: temperaturas máxima y
620         mínima y ocurrencia y montos de precipitación.
621     • models_data: datos usados efectivamente usados para ajustar los modelos (sin NAs)
622     • models_residuals: residuos de cada uno de los modelos. Es decir, la diferencia entre
623         el valor ajustado por el modelo (clima local) y el valor observado en el día i
624     • statistics_threshold: umbrales de amplitud térmica diaria por mes. Si la amplitud
625         simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de
626         mantener la consistencia entre variables
627     • exec_times: tiempo de ejecución de cada una de las etapas del ajuste

628 Cada uno de los GAMs ajustados se almacenan en el objeto gamgen_fit y pueden ser
629 evaluados con la función summary().

```

```

summary(gamgen_fit$fitted_models$`87448`$tmax_fit)

630 ##
631 ## Family: gaussian
632 ## Link function: identity
633 ##
634 ## Formula:
635 ## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +

```

```

636 ##      s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30) +
637 ##      s(SX1, SN1, k = 20) + s(SX2, SN2, k = 20) + s(SX3, SN3, k = 20) +
638 ##      s(SX4, SN4, k = 20)
639 ##
640 ## Parametric coefficients:
641 ##              Estimate Std. Error t value Pr(>|t|)
642 ## (Intercept) 25.62116    0.03189   803.4   <2e-16 ***
643 ## ---
644 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
645 ##
646 ## Approximate significance of smooth terms:
647 ##              edf Ref.df      F p-value
648 ## s(tmax_prev,tmin_prev) 29.9069 38.405 225.74   <2e-16 ***
649 ## s(prcp_occ)            0.9989  1.000 930.30   <2e-16 ***
650 ## s(prcp_occ_prev)       0.9995  1.000 2248.11   <2e-16 ***
651 ## s(doy)                 12.6336 28.000  73.12   <2e-16 ***
652 ## s(SX1,SN1)             3.0997  3.689  55.32   <2e-16 ***
653 ## s(SX2,SN2)             2.0001  2.000  89.25   <2e-16 ***
654 ## s(SX3,SN3)             4.4053  5.853  35.79   <2e-16 ***
655 ## s(SX4,SN4)             2.0001  2.000  86.85   <2e-16 ***
656 ## ---
657 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
658 ##
659 ## R-sq.(adj) =  0.713  Deviance explained = 71.4%
660 ## fREML =  58829  Scale est. = 13.964    n = 21466

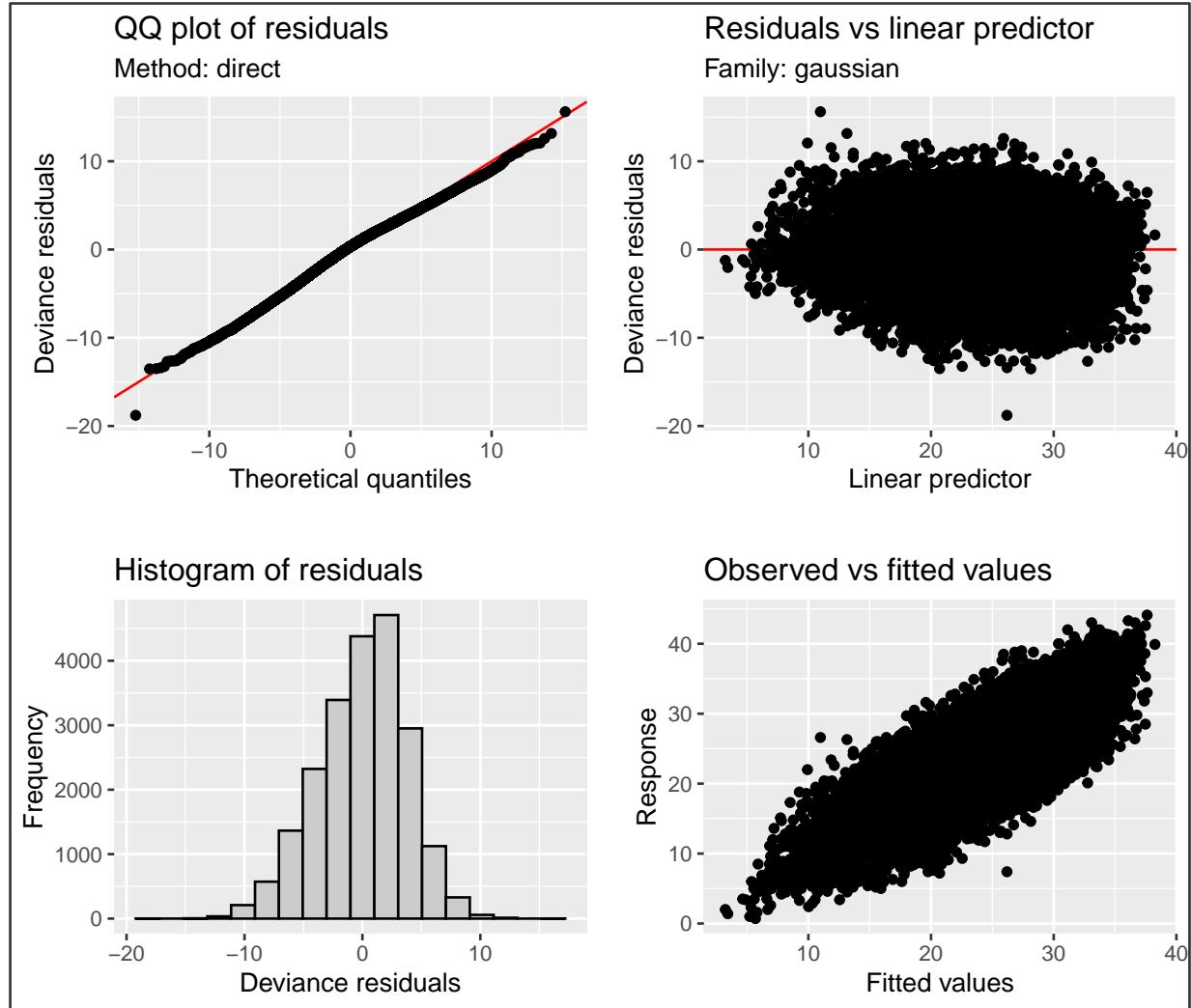
```

661 La función `summary` permite analizar los resultados del ajuste de cada uno de los modelos.  
662 Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la

663 parte superior bajo el apartado **Formula** y la significancia de cada uno de los términos  
664 del modelo en la tabla inmediatamente inferior. En este configuración, al incluirle variables  
665 estacionales, se incorporan nuevos términos el modelo como SX1, SX2, SX3, SX4 y SN1, SN2,  
666 SN3, SN4, que corresponden a variables dummy de las medias trimestrales de temperatura  
667 máxima y mínima, respectivamente. Se puede observar que todos los términos son altamente  
668 significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la  
669 varianza explicada por el modelo y el valor de R-ajustado.

670 El paquete **gratia** (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes  
671 de manera gráfica. Esta función toma los diagnósticos por defecto de la función **gam.check**  
672 del paquete **mgcv** (Wood, Simon (2001)) y los convierte en gráficos de la librería **ggplot2**  
673 (Wickham, Hadley(2011)) que son visualmente muy atractivos. La figura está dividida en  
674 cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo  
675 se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los  
676 datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior  
677 derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los  
678 residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones  
679 en los residuos indicaría que el modelo no ha explicado algún componente importante de  
680 la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los  
681 residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El  
682 último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores  
683 ajustados y observados.

```
gratia::appraise(gamgen_fit$fitted_models`87448`$tmax_fit) +  
  ggplot2::theme_bw()
```



684

685 Estos diagnósticos exploratorios pueden aplicarse a cada uno de los modelos ajustados por la  
686 función `local_calibrate`.

687 Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar  
688 series sintéticas para la o las estaciones usadas para calibrar el generador.

689 **5.3.3.2 Generación de series** La generación de series sigue la misma estructura ante-  
690 rior, una función para configurar la generación y otra que realiza la generación propiamente  
691 dicha.

692 Los argumentos de la función de control son:

- **nsim**: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o igual a 1.
- **seed**: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo porque se guarda junto a los resultados.
- **avbl\_cores**: cantidad de núcleos disponibles para la paralelización.
- **use\_spatially\_correlated\_noise**: utilizar la generación estocástica espacialmente correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible calcular los variogramas necesarios para la generación de los campos aleatorios. Se debe introducir un **boolean** (TRUE or FALSE).
- **use\_temporary\_files\_to\_save\_ram**: si se simulan muchas realizaciones o los recursos informáticos son escasos, esta opción permite guardar los resultados de cada una de las realizaciones en el disco liberando memoria RAM que quedará disponible para generar nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno único. Se debe introducir un **boolean** (TRUE or FALSE).
- **use\_temporary\_files\_to\_save\_ram**: esta opción permite eliminar los archivos temporales creados para ahorrar RAM luego de terminar la generación de todas las simulaciones. Se debe introducir un **boolean** (TRUE or FALSE).

```
control_sim <- gamwgen::local_simulation_control(
  nsim = 10, # Cantidad de simulaciones a realizar
  seed = 1234, # Semilla para que los resultados sean reproducibles
  avbl_cores = 6, # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = FALSE, # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = FALSE, # Guardar resultados intermedios para ahorrar espacio
  remove_temp_files_used_to_save_ram = TRUE) # Borrar los resultados intermedios creados
```

711 Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de

712 simulación son:

- 713 • **model**: objeto con el resultado de la función `local_calibrate()`
- 714 • **simulation\_locations**: objeto tipo `sf` con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- 718 • **start\_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato `date`
- 722 • **end\_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato `date`
- 726 • **control**: objeto de control creado con la función `control_sim()`.
- 727 • **output\_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- 729 • **output\_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- 731 • **seasonal\_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá un error. Se debe introducir un data frame con los valores agregados para las tres variables (precipitación y temperaturas máxima y mínima) pero no necesariamente deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma longitud que el período a generar

- 739 • `verbose`: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para cada una de las estaciones
# En este caso, por cuestiones de tiempo, vamos a cargar un objeto
# con los resultados de la simulación

simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2010-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
  # Covariables estacionales
  verbose = FALSE)

# Impresión de mensajes en la consola
```

- 740 Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y  
741 los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```
# Copiamos el archivo preajustado a nuestro directorio de trabajo
```

```

if (!fs::file_exists('output_data/local/simulated_local_conditional.RData')) {
  fs::file_copy(system.file('/autorun/local', "simulated_local_conditional.RData",
                           package = "gamwgen"),
               new_path = 'output_data/local/simulated_local_conditional.RData')
}

# Cargamos el archivo recientemente creado
load('output_data/local/simulated_local_conditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

742 ## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

743 ## [1] "nsim"
744 ## [2] "seed"
745 ## [3] "realizations_seeds"
746 ## [4] "simulation_points"
747 ## [5] "output_file_with_results"
748 ## [6] "output_file_fomart"
749 ## [7] "rdata_file_with_fitted_stations_and_climate"
750 ## [8] "exec_times"

751 La lista contiene los siguientes objetos:
    • nsim: cantidad de realizaciones.

```

- 753     • `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la  
 754       función de control.
- 755     • `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar  
 756       los resultados.
- 757     • `simulation_points`: puntos donde se generaron las series sintéticas.
- 758     • `output_file_with_results`: nombre del archivo con los resultados.
- 759     • `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- 760     • `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos me-  
 761       teorológicos observados que fueron utilizados en el ajuste. También se incluyen los  
 762       metadatos de cada uno de esos puntos.
- 763     • `exec_times`: tiempo de ejecución de la generación.

764 Ahora veremos el formato del archivo de salida que contiene las series sintéticas.

```
# Se carga el set de datos simulados
simulated_climate <- readr::read_csv(
  here::here('output_data/local/simulated_local_conditional.csv'))

# Primeras filas del objeto de salidas
knitr::kable(simulated_climate[1:10,])
```

realization	station_id	point_id	longitude	latitude	date	tmax	tmin	prcp	
1	87448	1	5001636	6256577	2010-01-01	28.05965	17.69135	7.534622	
1	87448	1	5001636	6256577	2010-01-02	27.32783	16.20019	0.000000	
1	87448	1	5001636	6256577	2010-01-03	35.01025	13.35766	0.000000	
1	87448	1	5001636	6256577	2010-01-04	34.18990	18.04710	0.000000	
765	1	87448	1	5001636	6256577	2010-01-05	30.59661	20.62692	0.000000
1	87448	1	5001636	6256577	2010-01-06	37.15162	17.44571	0.000000	
1	87448	1	5001636	6256577	2010-01-07	35.65166	18.36680	0.000000	
1	87448	1	5001636	6256577	2010-01-08	28.28271	13.77686	0.000000	
1	87448	1	5001636	6256577	2010-01-09	29.66045	13.73328	0.000000	
1	87448	1	5001636	6256577	2010-01-10	28.07379	14.17866	11.043809	

766 Al utilizar totales trimestrales de precipitación y medias trimestrales de temperaturas máx-  
 767 ima y mínima, las series generadas capturan los variaciones de baja frecuencia que se observan  
 768 en la serie histórica. A continuación se muestra una Figura que ilustra lo anterior para la  
 769 temperatura máxima.

```
ggplot2::ggplot() +  

  ggplot2::geom_boxplot(data = simulated_climate %>%  

    dplyr::mutate(month = lubridate::month(date),  

      year = lubridate::year(date),  

      date = as.Date(paste0(year, '-1', month, '-1', 1)))  

    ggplot2::aes(x = date,  

      y = tmax,  

      group = date,  

      fill = 'DarkOrange'),  

    alpha = 0.1) +  

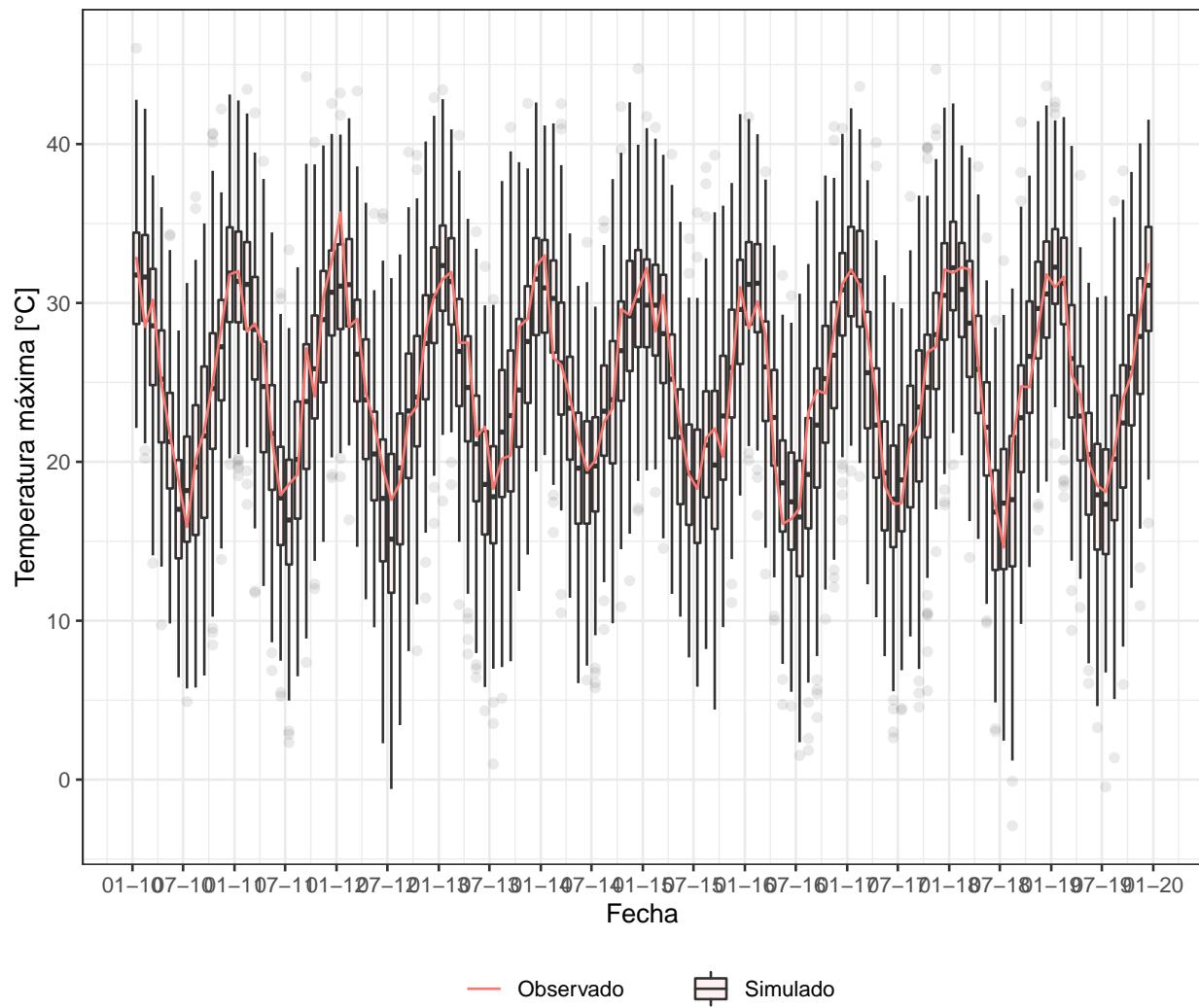
  ggplot2::geom_line(data = climate %>%
```

```

dplyr::filter(date > as.Date('2010-01-01')) %>%
dplyr::mutate(month = lubridate::month(date),
              year = lubridate::year(date)) %>%
dplyr::group_by(year, month) %>%
dplyr::summarise(tmax = median(tmax, na.rm = TRUE)) %>%
dplyr::mutate(date = as.Date(paste0(year, '-',
                                         month, '-',
                                         15)))
ggplot2::aes(
  x = date,
  y = tmax,
  group = 1,
  color = 'DarkOrange')) +
ggplot2::theme_bw() +
ggplot2::labs(x = 'Fecha', y = 'Temperatura máxima [°C]') +
ggplot2::scale_x_date(breaks = '6 months',
                      labels = scales::date_format("%m-%y")) +
ggplot2::scale_fill_discrete(name = "", labels = c("Simulado")) +
ggplot2::scale_color_discrete(name = "", labels = c("Observado")) +
ggplot2::theme(legend.position = 'bottom')

770 ## `summarise()` regrouping output by 'year' (override with `^.groups` argument)

```



771

772 Las cajas corresponde a las distintas realizaciones agregadas a escala mensual y la línea  
773 naranja corresponde a la temperatura media mensual calculada para los años 2010 a 2019.  
774 Se observa como las cajas suben y bajan al ritmo de la media observada y como capturan  
775 los pequeños cambios que ocurren en un año específico pero no en otros.

### 776 5.3.4 Series sintéticas correlacionadas espacialmente

777 Una tercera alternativa para la generación de datos sobre estaciones meteorológicas com-  
778 bina las antes mostradas pero generando el tiempo local con métodos que contemplan la  
779 autocorrelación espacial. Para utilizar esta alternativa se deben disponer de más de una

780 sola estación porque de otro modo no se podrían calcular los parámetros de los variogramas  
781 necesarios para la generación del tiempo local. Mientras más estaciones haya mejor, pero si  
782 pueden generar campos espaciales confiables con alrededor de 10 puntos.

783 **5.3.4.1 Crear archivos de entrada** Para este ejemplo se utilizan datos de varias esta-  
784 ciones pero el formato de los mismos es igual a los mostrados anteriormente.

785 Los archivos necesarios son:

- 786 • stations.csv  
787 • climate.csv

788 Los datos meteorológicos se dividen en dos archivos separados: **stations.csv** y  
789 **climate.csv**. Los nombres de los mismos no deben ser necesariamente iguales a los  
790 usados aquí.

791 Los metadatos de las estaciones se alojan en el archivo **stations.csv**. Este archivo contiene  
792 la información de las estaciones meteorológicas que serán usadas en el ajuste del modelo.  
793 Las variables que deben ser incluidas en la tabla son:

- 794 • station\_id: número único para cada estación meteorológica. La variable debe ser  
795 de tipo *integer*  
796 • latitude: latitud en grados decimales. La variable debe ser de tipo *double*  
797 • longitude: longitud en grados decimales. La variable debe ser de tipo *double*

798 La tabla puede tener más variables pero sólo se necesitan las anteriores.

799 A continuación se muestran la primera fila del dataset y los tipos de datos de cada una de  
800 las variables.

```
# Vista de los metadatos de la estación
knitr::kable(stations)
```

station_id	nombre	lat_dec	lon_dec	elev
86330	Artigas	-30.4000	-56.5100	120.38
86350	Rivera	-30.9000	-55.5400	241.94
86360	Salto	-31.4300	-57.9800	41.00
86430	Paysandú	-32.3500	-58.0400	61.12
86440	Melo	-32.3700	-54.1900	100.36
86460	Paso de los Toros	-32.8000	-56.5300	75.48
86490	Mercedes	-33.2500	-58.0700	17.01
86560	Colonia	-34.4500	-57.7700	18.00
86565	Rocha	-34.4900	-54.3100	18.16
86580	Aeropuerto Carrasco	-34.8300	-56.0100	32.88
90000001	Las Brujas	-34.6719	-56.3400	32.00
90000002	La Estanzuela	-34.3372	-57.6922	72.00
90000003	Tacuarembó	-31.7089	-55.8267	115.00
90000004	Treinta y Tres	-33.2750	-54.1722	22.00
90000005	Salto Grande	-31.2728	-57.8908	47.00

802 El objeto **stations** debe ser convertido de *tibble* a *sf*. El sistema de referencia espacial debe  
 803 ser planar. No es necesario un sistema de referencia espacial en particular, solamente las  
 804 coordenadas deben estar expresadas en metros.

```
# Convertimos el objeto stations a sf y se convierte su proyección de WGS 1984 a
# UTM Zona 21 S

stations %<>%
  sf::st_as_sf(coords = c("lon_dec", "lat_dec"), crs = 4326) %>%
  sf::st_transform(crs = 32721)
```

805 En el siguiente mapa muestra la distribución de las estaciones meteorológicas usadas en el  
 806 ejemplo.

```

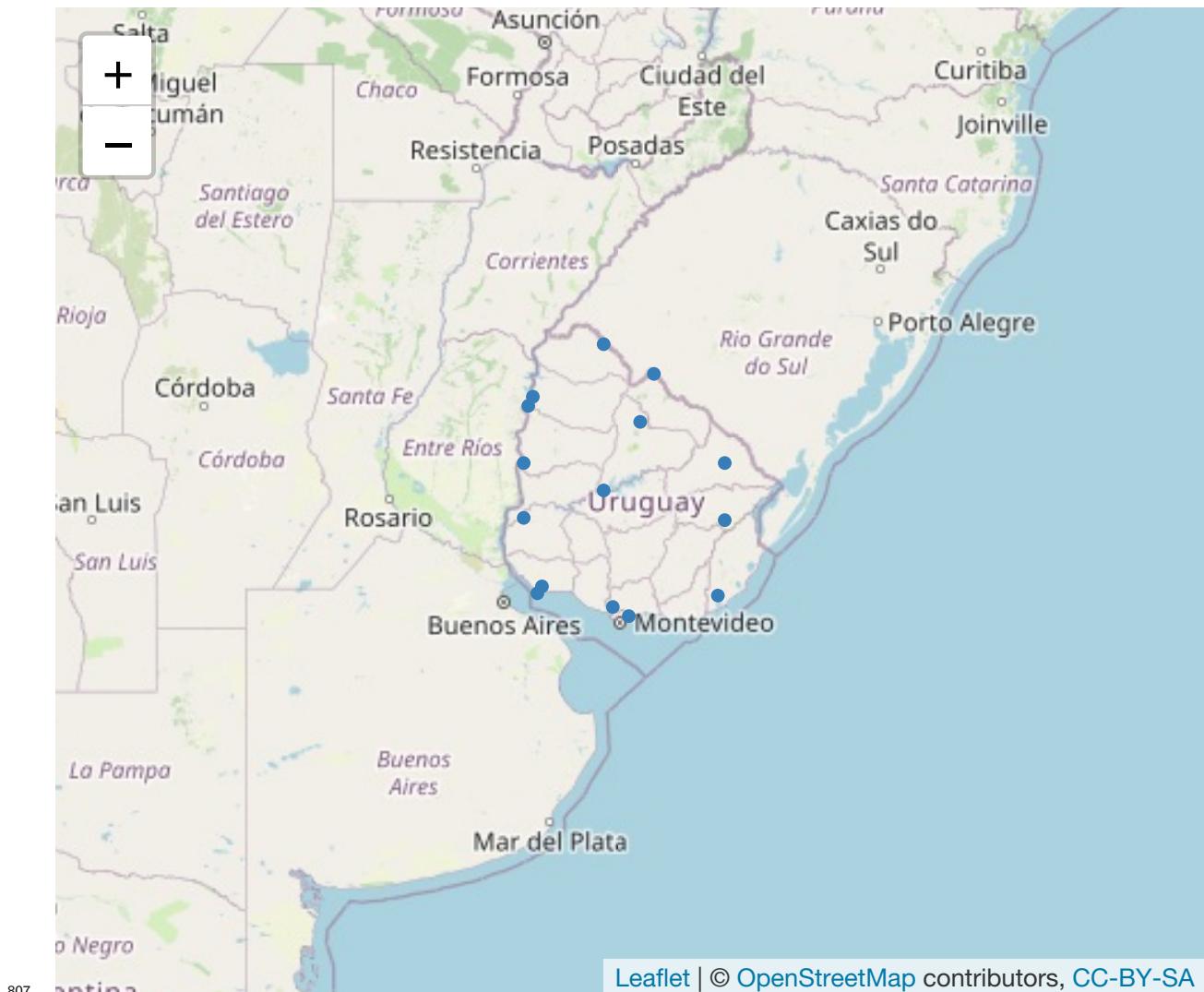
# Convertir el objeto con las estaciones a coordenadas geográficas

stations_geo <- stations %>%
  sf::st_transform(crs = 4326) %>%
  dplyr::mutate(lat = sf::st_coordinates(.)[,2],
               lon = sf::st_coordinates(.)[,1])

# Creación del mapa con la distribución de las estaciones

leaflet::leaflet(data = stations_geo) %>%
  leaflet::addTiles() %>%
  leaflet::setView(lat = -33, lng = -56, zoom = 5) %>%
  leaflet::addCircleMarkers(lat = ~lat, lng = ~lon, radius = 3,
                           fillColor = "#377eb8",
                           color = "#377eb8",
                           stroke = FALSE, fillOpacity = 1, opacity = 1,
                           popup = ~sprintf("<b>%s (%d)</b><br>Lat.: %.3f<br>Lon.: %.
                           nombre, station_id, lat, lon, elev))

```



807 La información climática se aloja en el archivo `climate.csv`. Este archivo contiene los datos  
 809 de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables que  
 810 deben ser incluidas en la tabla son:

- 811     • `date`: fecha del dato. La variable debe ser de tipo `date`
- 812     • `station_id`: número único para cada estación meteorológica. La variable debe ser  
       813        de tipo `integer`
- 814     • `prcp`: datos diarios de precipitación La variable debe ser de tipo `double`
- 815     • `tmax`: datos diarios de temperatura máxima. La variable debe ser de tipo `double`
- 816     • `tmin`: datos diarios de temperatura mínima. La variable debe ser de tipo `double`

817 A continuación se muestran las primeras cinco filas del dataset y los tipos de datos de cada  
818 una de las variables.

```
knitr::kable(climate[1:10,])
```

date	station_id	tmax	tmin	prcp
1981-01-01	86330	34.6	21.7	0.0
1981-01-02	86330	33.8	21.2	19.5
1981-01-03	86330	28.3	19.4	0.0
1981-01-04	86330	30.3	17.4	0.0
819 1981-01-05	86330	33.4	17.4	21.0
1981-01-06	86330	32.8	20.4	8.0
1981-01-07	86330	30.6	17.9	0.0
1981-01-08	86330	30.2	19.1	0.0
1981-01-09	86330	31.3	20.2	0.0
1981-01-10	86330	30.2	20.6	0.0

820 En total se utilizan 15 estaciones meteorológicas, 10 provistas por INUMET (Instituyo  
821 Uruguayo de Meteorología) y 5 por INIA (Instituto Nacional de Investigación Agropecuaria).

```
unique(climate$station_id)
```

```
822 ## [1] 86330    86350    86360    86430    86440    86460    86490    86560  
823 ## [9] 86565    86580 90000001 90000002 90000003 90000004 90000005
```

824 **5.3.4.2 Ajuste de los modelos** El ajuste de los modelos es igual que para los dos  
825 ejemplos antes mostrados. Se utiliza la función `control_fit` para la configuración del ajuste  
826 y `local_calibrate` para realizar el ajuste.

```

control_fit <- gamwgen::local_fit_control(
  prcp_occurrence_threshold = 0.1, # Umbral para la definición de días húmedos
  avbl_cores = 6, # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 32721) # Sistema de referencia espacial (en metros)

```

- 827 Este ejemplo se realizará utilizando covriables trimestrales pero es válido para series esta-  
 828 cionarias.

*# Agregación de valores diarios*

```
seasonal_covariates <- gamwgen::summarise_seasonal_climate(climate, umbral_faltantes =
```

*# Se muestran las primeras cinco filas*

```
knitr::kable(seasonal_covariates[1:10,])
```

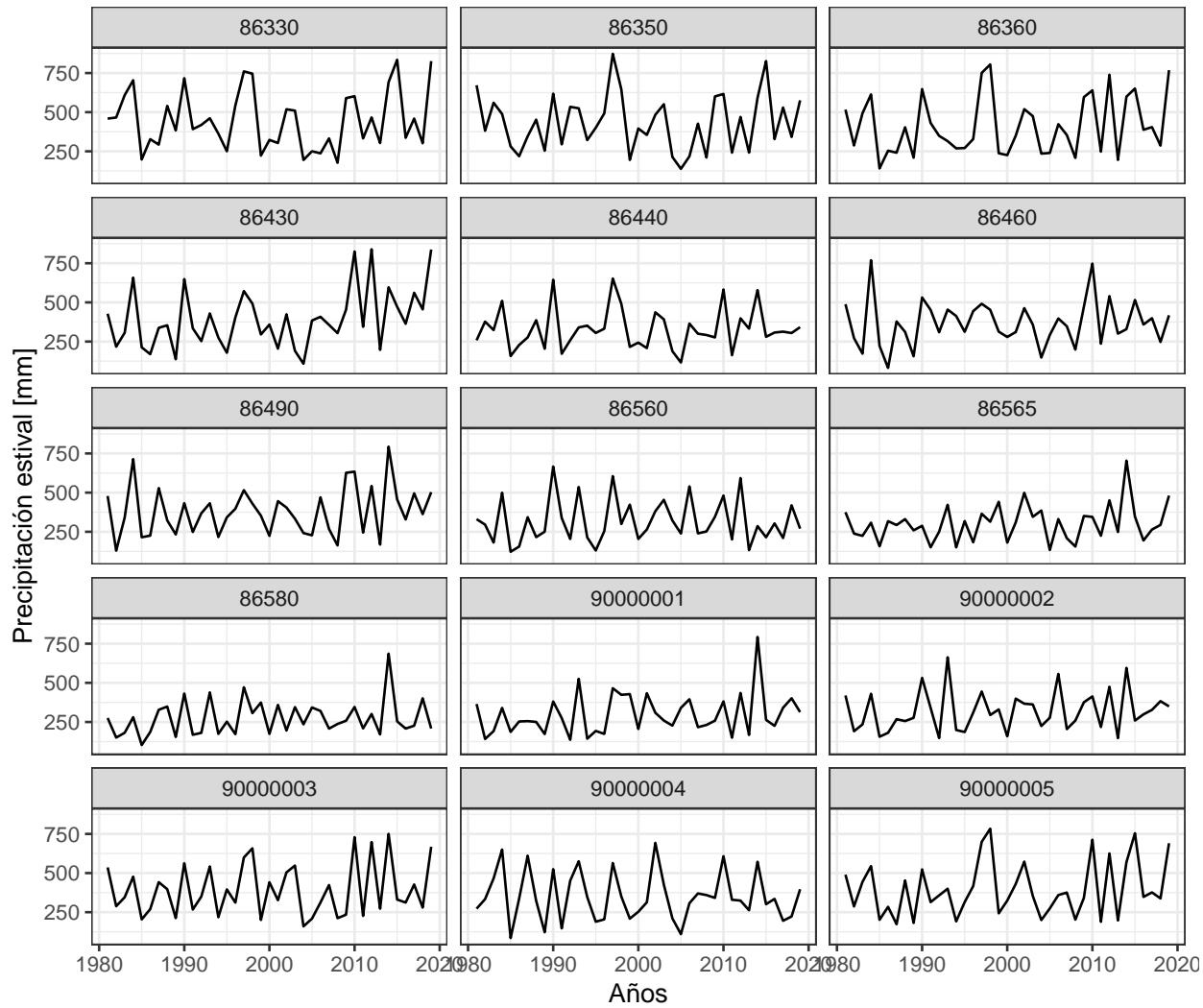
station_id	year	season	seasonal_prcp	seasonal_tmax	seasonal_tmin
86330	1981	1	458.9	30.32778	19.151111
86330	1981	2	370.4	26.44674	14.951087
86330	1981	3	211.2	19.76630	9.120652
86330	1981	4	272.0	24.73516	13.093407
829	86330	1982	1	465.9	30.33667
	86330	1982	2	229.0	26.74130
	86330	1982	3	365.4	19.48913
	86330	1982	4	651.7	24.68242
	86330	1983	1	609.0	31.39222
86330	1983	2	471.2	23.80761	13.631522

- 830 En la siguiente Figura se muestra la variación de la precipitación acumulada estival para las  
 831 15 estaciones utilizadas.

```

ggplot2::ggplot(data = seasonal_covariates %>%
  dplyr::filter(season == 1),
  ggplot2::aes(x = year, y = seasonal_prcp)) +
  ggplot2::geom_line() +
  ggplot2::facet_wrap(~station_id, ncol = 3) +
  ggplot2::theme_bw() +
  ggplot2::labs(x = 'Años', y = 'Precipitación estival [mm]')

```



832

833 Se observa en la figura que si bien hay cierta coincidencia, no todos los picos y valles ocurren  
 834 en el mismo momento. Utilizar un método que genere el tiempo local considerando estas  
 835 variaciones espaciales es muy útil para representar fehacientemente los patrones espaciales y

836 temporales.

```
# Al correr la función se realiza el ajuste de los cuatro modelos
# para cada una de las estaciones. En este caso, por cuestiones de tiempo a cargar un
# Si el usuario desea correrlo deberá ver la nota anterior.

gamgen_fit <- gamwgen::local_calibrate(
  climate = climate,
  # Registro histórico de variables meteorológicas
  stations = stations,
  # Estaciones meteorológicas
  seasonal_covariates = seasonal_covariates,
  # Totales trimestrales de precipitación
  control = control_fit,
  # Objeto de control
  verbose = FALSE)

# Impresión de mensajes en la consola.
```

837 Para esta demostración, cargamos el objeto con el ajuste del modelo ya realizado.

```
# Copiamos el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('input_data/local/fit_local_correlated_weather.RData')) {
  fs::file_copy(system.file('/autorun/local', "fit_local_correlated_weather.RData",
                           package = "gamwgen"),
               new_path = 'input_data/local/fit_local_correlated_weather.Rdata')
}

# Cargamos el archivo recientemente creado
load('input_data/local/fit_local_correlated_weather.RData')
```

```

# Clase del objeto con el ajuste del generador

class(gamgen_fit)

838 ## [1] "gamwgen"

# Contenido del modelo

names(gamgen_fit)

839 ## [1] "control"           "stations"          "climate"
840 ## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"
841 ## [7] "fitted_models"       "models_data"        "models_residuals"
842 ## [10] "statistics_threshold" "exec_times"

```

843 Dentro del objeto se guardan todo lo necesario para la simulación así como información  
 844 accesoria.

- 845 • **control**: copia de la configuración usada para calibrar el generador
- 846 • **stations**: estaciones meteorológicas utilizadas para la calibración
- 847 • **climate**: datos climáticos de cada uno de las estaciones
- 848 • **seasonal\_covariates**: series temporales de totales trimestrales de precipitación y  
 849 medias trimestrales de temperaturas máxima y mínima.
- 850 • **crs\_used\_to\_fit**: sistema de referencia espacial usado para proyectar
- 851 • **start\_climatology**: climatología diaria de cada una de las variables de entrada.
- 852 • **fitted\_models**: modelos ajustados, uno para cada variable: temperaturas máxima y  
 853 mínima y ocurrencia y montos de precipitación.
- 854 • **models\_data**: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- 855 • **models\_residuals**: residuos de cada uno de los modelos. Es decir, la diferencia entre  
 856 el valor ajustado por el modelo (clima local) y el valor observado en el día **i**

```

857 • statistics_threshold: umbrales de amplitud térmica diaria por mes. Si la amplitud
858 simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de
859 mantener la consistencia entre variables
860 • exec_times: tiempo de ejecución de cada una de las etapas del ajuste Al inspeccionar
861 el objeto con los resultados se puede ver que cada estación meteorológica tiene su
862 propia sublista donde se almacenan los modelos para cada una de ellas.

```

```
names(gamgen_fit$fitted_models)
```

```

863 ## [1] "86330"      "86350"      "86360"      "86430"      "86440"      "86460"
864 ## [7] "86490"      "86560"      "86565"      "86580"      "90000001"  "90000002"
865 ## [13] "90000003"  "90000004"  "90000005"

```

866 Cada uno de los GAMs ajustados se almacenan en el objeto `gamgen_fit` y pueden ser  
867 evaluados con la función `summary()`.

```
summary(gamgen_fit$fitted_models$`86330`$tmax_fit)
```

```

868 ##
869 ## Family: gaussian
870 ## Link function: identity
871 ##
872 ## Formula:
873 ## tmax ~ s(tmax_prev, tmin_prev, k = 50) + s(prcp_occ, bs = "re") +
874 ##       s(prcp_occ_prev, bs = "re") + s(doy, bs = "cc", k = 30) +
875 ##       s(SX1, SN1, k = 20) + s(SX2, SN2, k = 20) + s(SX3, SN3, k = 20) +
876 ##       s(SX4, SN4, k = 20)
877 ##
878 ## Parametric coefficients:

```

```

879 ##             Estimate Std. Error t value Pr(>|t|)

880 ## (Intercept) 26.18758     0.03397    770.9   <2e-16 ***
881 ## ---
882 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
883 ## 

884 ## Approximate significance of smooth terms:

885 ##             edf Ref.df      F p-value
886 ## s(tmax_prev,tmin_prev) 32.4286 40.704 235.57 <2e-16 ***
887 ## s(prcp_occ)           0.9979  1.000 520.81 <2e-16 ***
888 ## s(prcp_occ_prev)      0.9987  1.000 866.16 <2e-16 ***
889 ## s(doy)                10.3038 28.000  33.61 <2e-16 ***
890 ## s(SX1,SN1)            3.6312  4.458  26.92 <2e-16 ***
891 ## s(SX2,SN2)            2.0001  2.000  36.90 <2e-16 ***
892 ## s(SX3,SN3)            4.3491  5.631  21.09 <2e-16 ***
893 ## s(SX4,SN4)            2.0007  2.001  40.04 <2e-16 ***

894 ## ---
895 ## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
896 ## 

897 ## R-sq.(adj) =  0.778  Deviance explained = 77.9%
898 ## fREML = 35496  Scale est. = 8.8219    n = 14128

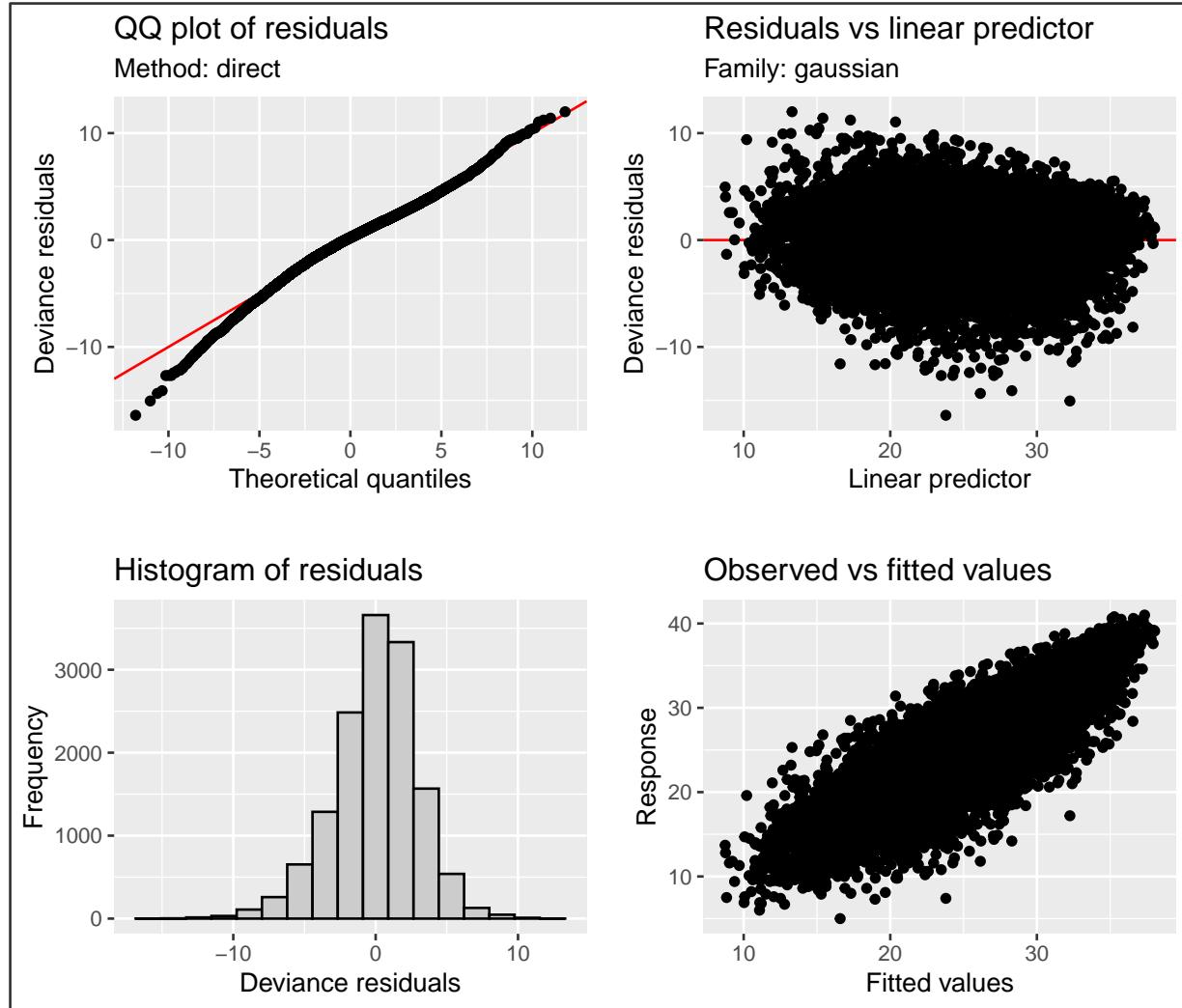
```

899 La función `summary` permite analizar los resultados del ajuste de cada uno de los modelos.  
900 Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la  
901 parte superior bajo el apartado `Formula` y la significancia de cada uno de los términos  
902 del modelo en la tabla inmediatamente inferior. En este configuración, al incluirle variables  
903 estacionales, se incorporan nuevos términos el modelo como SX1, SX2, SX3, SX4 y SN1, SN2,  
904 SN3, SN4, que corresponden a variables dummy de las medias trimestrales de temperatura  
905 máxima y mínima, respectivamente. Se puede observar que todos los términos son altamente

significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la varianza explicada por el modelo y el valor de R-ajustado.

El paquete `gratia` (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes de manera gráfica. Esta función toma los diagnósticos por defecto de la función `gam.check` del paquete `mgcv` (Wood, Simon (2001)) y los convierte en gráficos de la librería `ggplot2` (Wickham, Hadley(2011)) que son visualmente muy atractivos. La figura está dividida en cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones en los residuos indicaría que el modelo no ha explicado algún componente importante de la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores ajustados y observados.

```
gratia::appraise(gamgen_fit$fitted_models`^86330`$tmax_fit) +
  ggplot2::theme_bw()
```



922

923 Estos diagnósticos exploratorios pueden aplicarse a cada uno de los modelos ajustados por la  
924 función `local_calibrate`.

925 Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar  
926 series sintéticas para la o las estaciones usadas para calibrar el generador.

927 **5.3.4.3 Generación de series** La generación de series sigue la misma estructura ante-  
928 rior, una función para configurar la generación y otra que realiza la generación propiamente  
929 dicha.

930 Los argumentos de la función de control son:

- **nsim**: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o igual a 1.
- **seed**: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo porque se guarda junto a los resultados.
- **avbl\_cores**: cantidad de núcleos disponibles para la paralelización.
- **use\_spatially\_correlated\_noise**: utilizar la generación estocástica espacialmente correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible calcular los variogramas necesarios para la generación de los campos aleatorios. Se debe introducir un **boolean** (TRUE or FALSE).
- **use\_temporary\_files\_to\_save\_ram**: si se simulan muchas realizaciones o los recursos informáticos son escasos, esta opción permite guardar los resultados de cada una de las realizaciones en el disco liberando memoria RAM que quedará disponible para generar nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno único. Se debe introducir un **boolean** (TRUE or FALSE).
- **use\_temporary\_files\_to\_save\_ram**: esta opción permite eliminar los archivos temporales creados para ahorrar RAM luego de terminar la generación de todas las simulaciones. Se debe introducir un **boolean** (TRUE or FALSE).

```

control_sim <- gamwgen::local_simulation_control(
  nsim = 10,
  # Cantidad de simulaciones a realizar
  seed = 1234,
  # Semilla para que los resultados sean reproducibles
  avbl_cores = 6,
  # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = TRUE,
  # Usar modelo de ruido espacialmente correlacionado
)

```

```

use_temporary_files_to_save_ram = TRUE,
# Guardar resultados intermedios para ahorrar RAM
remove_temp_files_used_to_save_ram = TRUE)

# Borrar los resultados intermedios creados anteriormente

```

949 Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de  
 950 simulación son:

- 951 • **model**: objeto con el resultado de la función `local_calibrate()`
- 952 • **simulation\_locations**: objeto tipo `sf` con la ubicación de las estaciones a simular. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es necesario que todas estén presentes, se pueden generar series solo sobre algunas de ellas.
- 956 • **start\_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, ni tampoco posterior. Se debe introducir una fecha en formato **date**
- 960 • **end\_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron covariables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables, tampoco puede ser posterior. Se debe introducir una fecha en formato **date**
- 964 • **control**: objeto de control creado con la función `control_sim()`.
- 965 • **output\_folder**: ruta al directorio donde se guardarán los resultados, tanto finales como intermedios.
- 967 • **output\_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad, el archivo generado es un archivo de texto en formato separado por comas (.csv)
- 969 • **seasonal\_covariates**: datos agregados trimestrales. Si el ajuste se realizó con covariables, la generación también debe realizarse con ellas. Caso contrario se producirá

971 un error. Se debe introducir un data frame con los valores agregados para las tres  
972 variables (precipitación y temperaturas máxima y mínima) pero no necesariamente  
973 deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de  
974 algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar  
975 estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma  
976 longitud que el período a generar

- 977 • **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```
# Al correr la función se realiza la generación de series para
# cada una de las estaciones.

# En este caso, por cuestiones de tiempo, vamos a cargar un
# objeto con los resultados de la simulación

simulated_climate <- gamwgen::local_simulation(
  model = gamgen_fit,
  # Objeto con los resultados del ajuste
  simulation_locations = stations,
  # Estaciones para las cuales simular
  start_date = as.Date('2019-01-01'),
  # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-12-31'),
  # Fecha de fin de las simulaciones
  control = control_sim,
  # Objeto con la configuración
  output_folder = getwd(),
  # Directorio donde se guardarán los resultados
  output_filename = 'simulations.csv',
  # Nombre del archivo de salida
  seasonal_covariates = seasonal_covariates,
```

```

# Covariables estacionales
verbose = FALSE)

# Impresión de mensajes en la consola

```

978 Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y  
 979 los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```

# Copiamos el archivo preajustado a nuestro directorio de trabajo
if (!fs::file_exists('output_data/local/simulated_local_correlated_weather.RData')) {
  fs::file_copy(system.file('/autorun/local', "simulated_local_correlated_weather.RData",
                           new_path = 'output_data/local/simulated_local_correlated_weather.RData')
}

# Cargamos el archivo recientemente creado
load('output_data/local/simulated_local_unconditional.RData')

# Clase del objeto con el ajuste del generador
class(simulated_climate)

980 ## [1] "list"           "gamwgen.climate"

# Contenido del modelo
names(simulated_climate)

981 ## [1] "nsim"
982 ## [2] "seed"
983 ## [3] "realizations_seeds"
984 ## [4] "simulation_points"
985 ## [5] "output_file_with_results"
986 ## [6] "output_file_fomart"

```

```
987 ## [7] "rdata_file_with_fitted_stations_and_climate"  
988 ## [8] "exec_times"
```

989 La lista contiene los siguientes objetos:

- 990 • `nsim`: cantidad de realizaciones.
- 991 • `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la  
992 función de control.
- 993 • `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar  
994 los resultados.
- 995 • `simulation_points`: puntos donde se generaron las series sintéticas.
- 996 • `output_file_with_results`: nombre del archivo con los resultados.
- 997 • `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- 998 • `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos me-  
999 teorológicos observados que fueron utilizados en el ajuste. También se incluyen los  
1000 metadatos de cada uno de esos puntos.
- 1001 • `exec_times`: tiempo de ejecución de la generación.

1002 Ahora veremos el formato del archivo de salida que contiene las series sintéticas.

1003 Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones  
1004 que forman parte de `local_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen  
1005 los residuos y con la función `gamwgen:::generate_residuals_statistics` se calculan los  
1006 parámetros.

1007 En la tabla anterior se muestran los parámetros necesarios para generar el tiempo local de  
1008 las temperaturas máxima y mínima.

```
gen_noise_params <- gamwgen:::generate_month_params(  
  residuals = gamgen_fit$models_residuals,
```

```

observed_climate = gamgen_fit$models_data,
stations = gamgen_fit$stations)

# Ejemplo de variograma de residuos de temperatura máxima para días secos

gen_noise_params[[1]]$variogram_parameters$tmax_dry

1009 ## [1] 0.000000 6.978337 495.779099

```

1010 Para cada una de las variables, precipitación y temperaturas máxima y mínima, se ajusta  
 1011 un variograma por máxima verosimilitud que permite representar la variabilidad espacial  
 1012 de los residuos de los modelos GAMs, que corresponden al tiempo local. En el variograma  
 1013 se muestran los tres parámetros **nugget**, **psill** y **range**. El rango de los datos es de 500  
 1014 km, lo que es lógico si se considera que los puntos están distribuidos homogéneamente en la  
 1015 República Oriental del Uruguay.

1016 En la siguiente FIgura se muestra la variabilidad espacial del tiempo local de temperatura  
 1017 del día 1 de enero de 2019 para días secos. Cabe mencionar que se crean también los mismos  
 1018 datos para los días húmedos y en función de la ocurrencia de precipitación se selecciona uno  
 1019 u otro.

```
RandomFields::RFoptions(printlevel = 0, spConform = FALSE)
```

```

temp_local <- control_sim$temperature_noise_generating_function(simulation_points =
  dplyr::mutate(
    gen_noise_params
    month_number = 1L
    selector = 'Dry',
    seed = NULL) %>%

```

```

dplyr::mutate(date = as.Date(paste0('2019-', '01', '-', '01')))) %>%
tidyrr::gather(variable, valor, -c('date', 'geometry'))

# Mapa de Uruguay

uruguay <- raster::getData('GADM', country='URY', level=1) %>%
sf::st_as_sf(.) %>%
sf::st_transform(crs = 32721)

# Paleta de colores

colr <- grDevices::colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))

# Quiebres de la escala de colores

quiebres <- c(-8.0, -6.0, -4.0,-2.0, 0, 2.0, 4.0, 6.0, 8.0)

# Etiquetas para los colores

etiquetas <- c('-8', '-6', '-4', '-2', '0', '2', '4', '6', '8')

# Etiquetas de las facetas

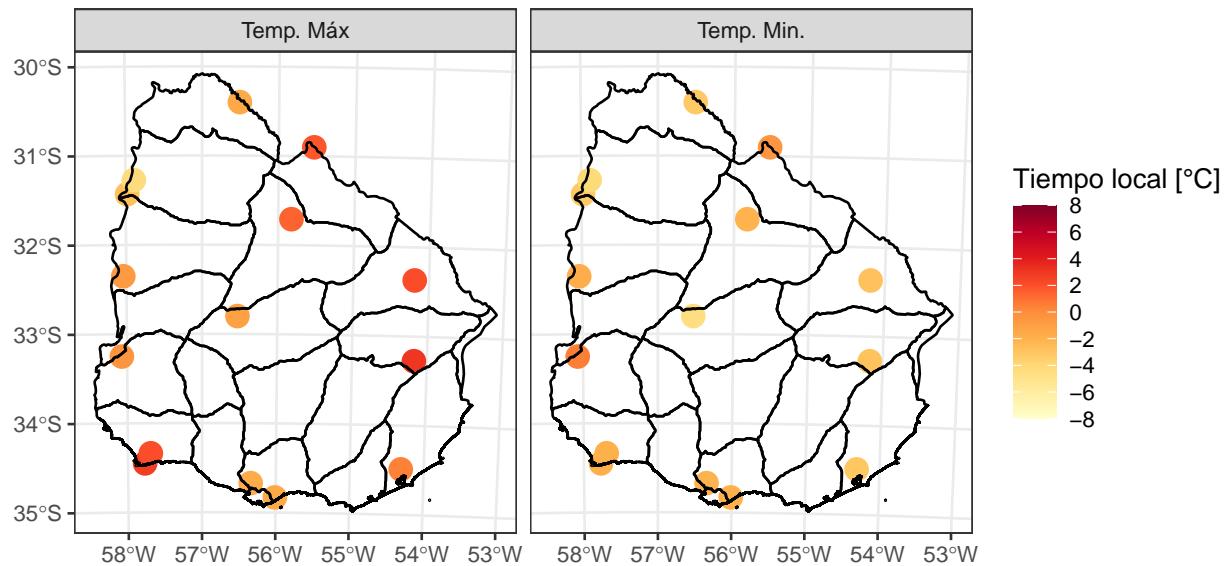
labs <- c("Temp. Máx", "Temp. Min.")

names(labs) <- c("tmax_residuals", "tmin_residuals")

ggplot2::ggplot(data = temp_local) +
ggplot2::geom_sf(ggplot2::aes(color = valor), size = 4) +
ggplot2::scale_color_gradientn(name = "Tiempo local [°C]",
                               colours = colr(9),
                               breaks = quiebres,
                               labels = etiquetas,
                               limits = c(-8, 8)) +
ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +

```

```
ggplot2::facet_wrap(~variable, labeller = ggplot2::labeller(variable = labs)) +
  ggplot2::theme_bw()
```



1020

1021 Se puede observar como el tiempo local tiene una estructura espacial que es representada a  
 1022 través del variograma y permite que los resultados para las distintas estaciones sean espa-  
 1023 cialmente consistentes. Esto quiere decir que estaciones meteorológicas cercanas tenderán a  
 1024 tener valores de tiempo local similares.

1025 **5.4 Generación de series sintéticas para una grilla regular**

1026 La generación de series sobre grillas regular es muy similar a lo mostrado anteriormente.  
1027 Los fundamentos son los mismos, sólo los modelos estadísticos que modelan el clima local  
1028 incorporan un nuevo término que permite incluir la dimensión espacial.

1029 **5.4.0.1 Crear archivos de entrada** Para este ejemplo se utilizan datos de varias esta-  
1030 ciones pero el formato de los mismos es igual a los mostrados anteriormente.

1031 Los archivos necesarios son:

- 1032     • stations.csv  
1033     • climate.csv

1034 Los datos meteorológicos se dividen en dos archivos separados: `stations.csv` y  
1035 `climate.csv`. Los nombres de los mismos no deben ser necesariamente iguales a los  
1036 usados aquí.

1037 Los metadatos de las estaciones se alojan en el archivo `stations.csv`. Este archivo contiene  
1038 la información de las estaciones meteorológicas que serán usadas en el ajuste del modelo.  
1039 Las variables que deben ser incluidas en la tabla son:

- 1040     • station\_id: número único para cada estación meteorológica. La variable debe ser  
1041         de tipo *integer*  
1042     • latitude: latitud en grados decimales. La variable debe ser de tipo *double*  
1043     • longitude: longitud en grados decimales. La variable debe ser de tipo *double*

1044 La tabla puede tener más variables pero sólo se necesitan las anteriores.

1045 A continuación se muestran la primera fila del dataset y los tipo de datos de cada una de  
1046 las variables.

```
# Vista de los metadatos de la estación
knitr::kable(stations)
```

station_id	nombre	lat_dec	lon_dec	elev
86330	Artigas	-30.4000	-56.5100	120.38
86350	Rivera	-30.9000	-55.5400	241.94
86360	Salto	-31.4300	-57.9800	41.00
86430	Paysandú	-32.3500	-58.0400	61.12
86440	Melo	-32.3700	-54.1900	100.36
86460	Paso de los Toros	-32.8000	-56.5300	75.48
86490	Mercedes	-33.2500	-58.0700	17.01
86560	Colonia	-34.4500	-57.7700	18.00
86565	Rocha	-34.4900	-54.3100	18.16
86580	Aeropuerto Carrasco	-34.8300	-56.0100	32.88
90000001	Las Brujas	-34.6719	-56.3400	32.00
90000002	La Estanzuela	-34.3372	-57.6922	72.00
90000003	Tacuarembó	-31.7089	-55.8267	115.00
90000004	Treinta y Tres	-33.2750	-54.1722	22.00
90000005	Salto Grande	-31.2728	-57.8908	47.00

1047  
1048 El objeto **stations** debe ser convertido de *tibble* a *sf*. El sistema de referencia espacial debe  
ser planar. No es necesario un sistema de referencia espacial en particular, solamente las  
1049 coordenadas deben estar expresadas en metros.  
1050

```
# Convertimos el objeto stations a sf y se convierte su proyección de WGS 1984 a
# UTM Zona 21 S
stations %<>%
sf::st_as_sf(coords = c("lon_dec", "lat_dec"), crs = 4326) %>%
sf::st_transform(crs = 32721)
```

- 1051 En el siguiente mapa muestra la distribución de las estaciones meteorológicas usadas en el  
1052 ejemplo.

```
# Convertir el objeto con las estaciones a coordenadas geográficas
stations_geo <- stations %>%
  sf:::st_transform(crs = 4326) %>%
  dplyr::mutate(lat = sf:::st_coordinates(.)[,2],
    lon = sf:::st_coordinates(.)[,1])

# Creación del mapa con la distribución de las estaciones
leaflet::leaflet(data = stations_geo) %>%
  leaflet::addTiles() %>%
  leaflet::setView(lat = -33, lng = -56, zoom = 5) %>%
  leaflet::addCircleMarkers(lat = ~lat, lng = ~lon, radius = 3,
    fillColor = "#377eb8",
    color = "#377eb8",
    stroke = FALSE, fillOpacity = 1, opacity = 1,
    popup = ~sprintf("<b>%s (%d)</b><br>Lat.: %.3f<br>Lon.: %.
      nombre, station_id, lat, lon, elev))
```



1054 La información climática se aloja en el archivo `climate.csv`. Este archivo contiene los datos  
 1055 de las estaciones meteorológicas que serán usadas en el ajuste del modelo. Las variables que  
 1056 deben ser incluidas en la tabla son:

- 1057     • `date`: fecha del dato. La variable debe ser de tipo `date`
- 1058     • `station_id`: número único para cada estación meteorológica. La variable debe ser  
               de tipo `integer`
- 1059     • `prcp`: datos diarios de precipitación La variable debe ser de tipo `double`
- 1060     • `tmax`: datos diarios de temperatura máxima. La variable debe ser de tipo `double`
- 1061     • `tmin`: datos diarios de temperatura mínima. La variable debe ser de tipo `double`
- 1062

1063 A continuación se muestran las primeras cinco filas del dataset y los tipos de datos de cada  
1064 una de las variables.

```
knitr::kable(climate[1:10,])
```

date	station_id	tmax	tmin	prcp	
1981-01-01	86330	34.6	21.7	0.0	
1981-01-02	86330	33.8	21.2	19.5	
1981-01-03	86330	28.3	19.4	0.0	
1981-01-04	86330	30.3	17.4	0.0	
<small>1065</small>	1981-01-05	86330	33.4	17.4	21.0
	1981-01-06	86330	32.8	20.4	8.0
	1981-01-07	86330	30.6	17.9	0.0
	1981-01-08	86330	30.2	19.1	0.0
	1981-01-09	86330	31.3	20.2	0.0
	1981-01-10	86330	30.2	20.6	0.0

1066 En total se utilizan 15 estaciones meteorológicas, 10 provistas por INUMET (Instituyo  
1067 Uruguayo de Meteorología) y 5 por INIA (Instituto Nacional de Investigación Agropecuaria).

```
unique(climate$station_id)
```

1068 ## [1] 86330 86350 86360 86430 86440 86460 86490 86560  
1069 ## [9] 86565 86580 90000001 90000002 90000003 90000004 90000005

1070 **5.4.0.2 Ajuste de los modelos** El ajuste de los modelos es igual que para los dos  
1071 ejemplos antes mostrados. Se utiliza la función `control_fit` para la configuración del ajuste  
1072 y `spatial_calibrate` para realizar el ajuste.

```

control_fit <- gamwgen::spatial_fit_control(
  prcp_occurrence_threshold = 0.1, # Umbral para la definición de días húmedos
  avbl_cores = 6, # Cantidad de núcleos disponibles
  planar_crs_in_metric_coords = 32721) # Sistema de referencia espacial (en metros)

1073 Este ejemplo se realizará utilizando covriables trimestrales pero es válido para series esta-
1074 cionarias.

```

*# Agregación de valores diarios*

```
seasonal_covariates <- gamwgen::summarise_seasonal_climate(climate, umbral_faltantes =
```

*# Se muestran las primeras cinco filas*

```
knitr::kable(seasonal_covariates[1:10,])
```

station_id	year	season	seasonal_prcp	seasonal_tmax	seasonal_tmin
86330	1981	1	458.9	30.32778	19.151111
86330	1981	2	370.4	26.44674	14.951087
86330	1981	3	211.2	19.76630	9.120652
86330	1981	4	272.0	24.73516	13.093407
1075	86330	1982	1	465.9	30.33667
	86330	1982	2	229.0	26.74130
	86330	1982	3	365.4	19.48913
	86330	1982	4	651.7	24.68242
	86330	1983	1	609.0	31.39222
	86330	1983	2	471.2	23.80761
					13.631522

*# Al correr la función se realiza el ajuste de los cuatro modelos para cada una de las estaciones. En este caso, por cuestiones de tiempo a cargar un objeto ya precalculado.*

*# Si el usuario desea correrlo deberá ver la nota anterior.*

```

gamgen_fit <- gamwgen::spatial_calibrate(climate = climate, # Registro histórico de var  

                                             stations = stations, # Estaciones meteorológicas  

                                             seasonal_covariates = seasonal_covariates, # Totales trimestrales de precipitación  

                                             control = control_fit, # Objeto de control  

                                             verbose = FALSE) # Impresión de mensajes en la consola.

```

1076 Para esta demostración, cargamos el objeto con el ajuste del modelo ya realizado.

```

# Copiamos el archivo preajustado a nuestro directorio de trabajo  

if (!fs::file_exists('input_data/spatial/fit_spatial_conditional.RData')) {  

  fs::file_copy(system.file('/autorun/spatial', "fit_spatial_conditional.RData", package = 'gamwgen'))  

  new_path = 'input_data/spatial/fit_spatial_conditional.Rdata'  

}  
  

# Cargamos el archivo recientemente creado  

load('input_data/spatial/fit_spatial_conditional.RData')  
  

# Clase del objeto con el ajuste del generador  

class(gamgen_fit)  
  

1077 ## [1] "gamwgen"  
  

# Contenido del modelo  

names(gamgen_fit)  
  

1078 ## [1] "control"           "stations"          "climate"  

1079 ## [4] "seasonal_covariates" "crs_used_to_fit"   "start_climatology"  

1080 ## [7] "fitted_models"      "models_data"       "models_residuals"  

1081 ## [10] "statistics_threshold" "exec_times"

```

1082 Dentro del objeto se guardan todo lo necesario para la simulación así como información  
1083 accesoria.

- 1084 • **control**: copia de la configuración usada para calibrar el generador
- 1085 • **stations**: estaciones meteorológicas utilizadas para la calibración
- 1086 • **climate**: datos climáticos de cada uno de las estaciones
- 1087 • **seasonal\_covariates**: series temporales de totales trimestrales de precipitación y  
1088 medias trimestrales de temperaturas máxima y mínima.
- 1089 • **crs\_used\_to\_fit**: sistema de referencia espacial usado para proyectar
- 1090 • **start\_climatology**: climatología diaria de cada una de las variables de entrada.
- 1091 • **fitted\_models**: modelos ajustados, uno para cada variable: temperaturas máxima y  
1092 mínima y ocurrencia y montos de precipitación.
- 1093 • **models\_data**: datos usados efectivamente usados para ajustar los modelos (sin NAs)
- 1094 • **models\_residuals**: residuos de cada uno de los modelos. Es decir, la diferencia entre  
1095 el valor ajustado por el modelo (clima local) y el valor observado en el día **i**
- 1096 • **statistics\_threshold**: umbrales de amplitud térmica diaria por mes. Si la amplitud  
1097 simulada está fuera de este rango, se repetirá la simulación para ese día a los fines de  
1098 mantener la consistencia entre variables
- 1099 • **exec\_times**: tiempo de ejecución de cada una de las etapas del ajuste

1100 A diferencia de la configuración que ajusta distintos modelos para cada estación meteorológi-  
1101 ca, en te caso se ajusta un sólo modelos para toda la región de interés. Al visualizar lo que  
1102 está almacenado en la sublista **fitted\_models** se ve que solo hay cuatro GAMs, dos para la  
1103 precipitación y dos para las temperaturas máxima y mínima.

```
names(gamgen_fit$fitted_models)
```

```
1104 ## [1] "prcp_occ_fit" "prcp_amt_fit" "tmax_fit"      "tmin_fit"
```

1105 Cada uno de los GAMs ajustados se almacenan en el objeto `gamgen_fit` y pueden ser  
1106 evaluados con la función `summary()`.

```
summary(gamgen_fit$fitted_models$tmax_fit)

1107 ##
1108 ## Family: gaussian
1109 ## Link function: identity
1110 ##
1111 ## Formula:
1112 ## tmax ~ te(tmax_prev, tmin_prev, longitude, latitude, d = c(2,
1113 ##           2), k = length(unique_stations)) + te(type_day, longitude,
1114 ##           latitude, d = c(1, 2), bs = c("re", "tp"), k = length(unique_stations)) +
1115 ##           te(type_day_prev, longitude, latitude, d = c(1, 2), bs = c("re",
1116 ##             "tp"), k = length(unique_stations)) + te(doy, longitude,
1117 ##             latitude, d = c(1, 2), bs = c("cc", "tp"), k = length(unique_stations)) +
1118 ##             te(SX1, SN1, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
1119 ##             te(SX2, SN2, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
1120 ##             te(SX3, SN3, longitude, latitude, d = c(2, 2), k = length(unique_stations)) +
1121 ##             te(SX4, SN4, longitude, latitude, d = c(2, 2), k = length(unique_stations))
1122 ##
1123 ## Parametric coefficients:
1124 ##                               Estimate Std. Error t value Pr(>|t|)
1125 ## (Intercept)      25.91      29.52   0.878    0.38
1126 ##
1127 ## Approximate significance of smooth terms:
1128 ##                                         edf Ref.df      F p-value
1129 ## te(tmax_prev,tmin_prev,longitude,latitude) 76.902  86.484 1267.40 <2e-16 ***
```

```

1130 ## te(type_day,longitude,latitude)           27.680 29.000 161.22 <2e-16 ***
1131 ## te(type_day_prev,longitude,latitude)       14.703 15.000 726.24 <2e-16 ***
1132 ## te(doy,longitude,latitude)                 47.907 195.000 77.44 <2e-16 ***
1133 ## te(SX1,SN1,longitude,latitude)            18.355 23.456 75.14 <2e-16 ***
1134 ## te(SX2,SN2,longitude,latitude)            6.005   6.009 184.32 <2e-16 ***
1135 ## te(SX3,SN3,longitude,latitude)            30.642 36.395 48.54 <2e-16 ***
1136 ## te(SX4,SN4,longitude,latitude)            6.002   6.004 193.99 <2e-16 ***
1137 ##
1138 ## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
1139 ##
1140 ## R-sq.(adj) = 0.779 Deviance explained = 77.9%
1141 ## fREML = 5.221e+05 Scale est. = 8.9846 n = 207269

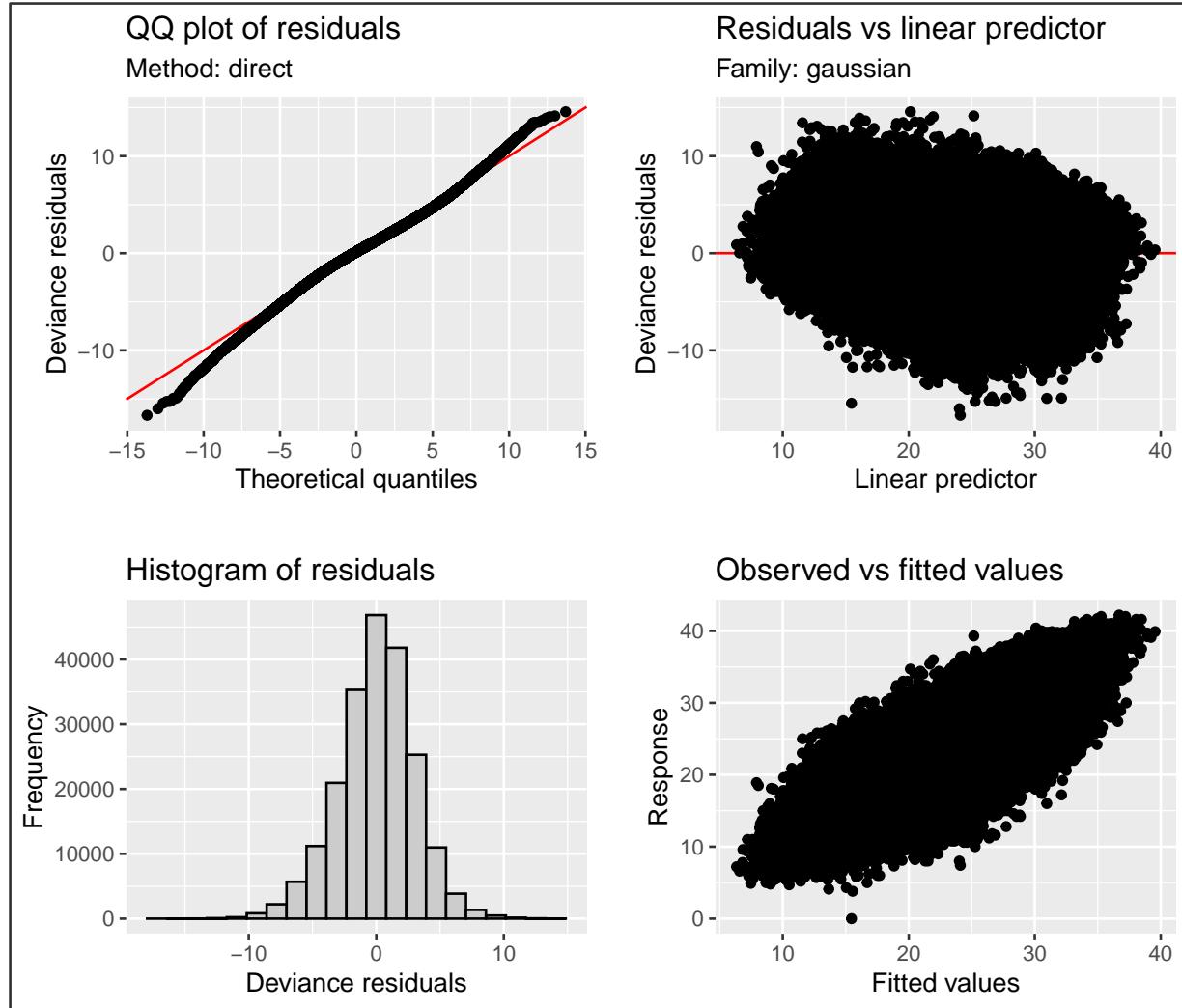
```

1142 La función **summary** permite analizar los resultados del ajuste de cada uno de los modelos.  
1143 Para el caso del modelo de temperatura máxima podemos ver la fórmula del GAM en la  
1144 parte superior bajo el apartado **Formula** y la significancia de cada uno de los términos  
1145 del modelo en la tabla inmediatamente inferior. En este configuración, al incluirle variables  
1146 estacionales, se incorporan nuevos términos el modelo como SX1, SX2, SX3, SX4 y SN1, SN2,  
1147 SN3, SN4, que corresponden a variables dummy de las medias trimestrales de temperatura  
1148 máxima y mínima, respectivamente. Se puede observar que todos los términos son altamente  
1149 significativos. También se incluyen como pruebas de bondad del ajuste el porcentaje de la  
1150 varianza explicada por el modelo y el valor de R-ajustado.

1151 El paquete **gratia** (Simpson, Gavin (2018)) es muy útil para la visualización de los ajustes  
1152 de manera gráfica. Esta función toma los diagnósticos por defecto de la función **gam.check**  
1153 del paquete **mgcv** (Wood, Simon (2001)) y los convierte en gráficos de la librería **ggplot2**  
1154 (Wickham, Hadley(2011)) que son visualmente muy atractivos. La figura está dividida en  
1155 cuatro paneles, cada uno mostrando un diagnóstico diferente. En el panel superior izquierdo

1156 se muestra un Q-Q Plot de los residuos. Si el modelo ha ajustado satisfactoriamente los  
1157 datos, los puntos deberían estar sobre la recta 1:1 demarcada en rojo. El panel superior  
1158 derecho muestra los residuos vs el predictor lineal. El objetivo de este diagnóstico es que los  
1159 residuos se distribuyan al azar y que no tengan un patrón claro. La presencia de patrones  
1160 en los residuos indicaría que el modelo no ha explicado algún componente importante de  
1161 la variabilidad de los datos. En el panel inferior izquierdo se muestra un histograma de los  
1162 residuos siendo deseable que lo mismos tengan una distribución próxima a la Normal. El  
1163 último gráfico en el panel inferior derecho muestra una gráfica de dispersión entre los valores  
1164 ajustados y observados.

```
gratia::appraise(gamgen_fit$fitted_models$tmax_fit) +  
  ggplot2::theme_bw()
```



1165

1166 Estos diagnósticos exploratorios pueden aplicarse a cada uno de los modelos ajustados por la  
1167 función `spatial_calibrate`.

1168 Con la creación del objeto `gamgen_fit` finaliza el proceso de ajuste y ya se pueden generar  
1169 series sintéticas para la o las estaciones usadas para calibrar el generador.

1170 **5.4.0.3 Generación de series** La generación de series sigue la misma estructura ante-  
1171 rior, una función para configurar la generación y otra que realiza la generación propiamente  
1172 dicha.

1173 Los argumentos de la función de control son:

- 1174     • **nsim**: cantidad de simulaciones a realizar. Se debe ingresar un valor **entero** mayor o  
 1175         igual a 1.
- 1176     • **seed**: semilla. Se debe ingresar cualquier numero **entero**. No es necesario recordarlo  
 1177         porque se guarda junto a los resultados.
- 1178     • **avbl\_cores**: cantidad de núcleos disponibles para la paralelización.
- 1179     • **use\_spatially\_correlated\_noise**: utilizar la generación estocástica espacialmente  
 1180         correlacionada. Esta opción sólo es válida si en el ajuste y en la simulación se usaron  
 1181         más de cinco estaciones meteorológicas diferentes. Con un menor número no es posible  
 1182         calcular los variogramas necesarios para la generación de los campos aleatorios. Se  
 1183         debe introducir un **boolean** (TRUE or FALSE).
- 1184     • **use\_temporary\_files\_to\_save\_ram**: si se simulan muchas realizaciones o los recursos  
 1185         informáticos son escasos, esta opción permite guardar los resultados de cada una de las  
 1186         realizaciones en el disco liberando memoria RAM que quedará disponible para generar  
 1187         nuevas simulaciones. Al finalizar la generación todos los archivos se combinan en uno  
 1188         único. Se debe introducir un **boolean** (TRUE or FALSE).
- 1189     • **use\_temporary\_files\_to\_save\_ram**: esta opción permite eliminar los archivos tem-  
 1190         porales creados para ahorrar RAM luego de terminar la generación de todas las simu-  
 1191         laciones. Se debe introducir un **boolean** (TRUE or FALSE).

```
control_sim <- gamwgen::spatial_simulation_control(
  nsim = 10, # Cantidad de simulaciones a realizar
  seed = 1234, # Semilla para que los resultados sean reproducibles
  avbl_cores = 6, # Cantidad de núcleos disponibles a utilizar
  use_spatially_correlated_noise = TRUE, # Usar modelo de ruido espacialmente correlacionado
  use_temporary_files_to_save_ram = TRUE, # Guardar resultados intermedios para ahorrar RAM
  remove_temp_files_used_to_save_ram = TRUE) # Borrar los resultados intermedios creados

# Agregación de valores diarios
```

```

seasonal_covariates <- gamwgen::summarise_seasonal_climate(climate, umbral_faltantes = 0)

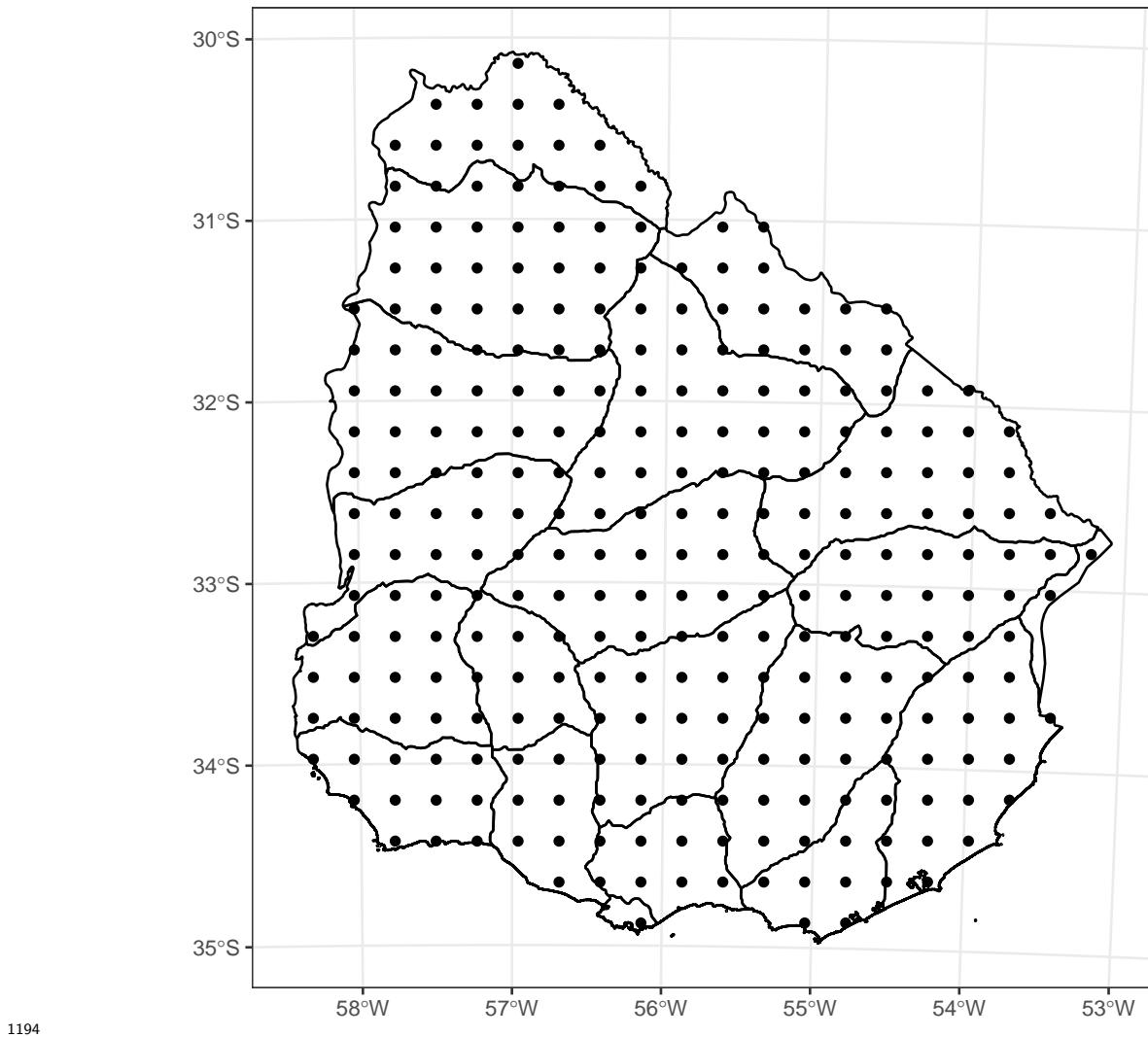
# Shapefile de Uruguay
uruguay <- raster::getData('GADM', country='URY', level=1) %>%
  sf::st_as_sf(.) %>%
  sf::st_transform(crs = 32721)

# Creación de la grilla sobre la que se simulará
grilla_simulacion_centers <- uruguay %>%
  dplyr::select(geometry) %>%
  sf::st_transform(gamgen_fit$crs_used_to_fit) %>%
  sf::st_make_grid(cellsize = 25000, what = 'centers') %>%
  sf::st_sf(grid_id = 1:length(.), geometry = .) %>%
  sf::st_intersection(uruguay)

## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries

ggplot2::ggplot() +
  ggplot2::geom_sf(data = grilla_simulacion_centers) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw()

```



1194

1195 Al utilizar covariables trimestrales es necesario que éstas se interpolen para cada uno de  
 1196 los puntos de la grilla. A continuación se muestran dos ejemplos, uno para precipitación  
 1197 acumulada para el trimestre estival de 2019 y la temperatura máxima media del mismo  
 1198 período.

```
simulation_dates <-
  tibble::tibble(date = seq.Date(from = as.Date('2019-01-01'),
                                 to = as.Date('2019-01-31'),
                                 by = "days")) %>%
  dplyr::mutate(year = lubridate::year(date),
```

```

month   = lubridate::month(date),
season = lubridate::quarter(date, fiscal_start = 12))

grilla_simulacion_centers <- uruguay %>%
sf::st_transform(gamgen_fit$crs_used_to_fit) %>%
sf::st_make_grid(cellsize = 25000, what = 'centers') %>%
sf::st_sf(grid_id = 1:length(.), geometry = .) %>%
sf::st_intersection(uruguay) %>%
dplyr::select(geometry) %>%
dplyr::mutate(latitude = sf::st_coordinates(.)[,2],
longitude = sf::st_coordinates(.)[,1])

seasonal_covariates <-
gamwgen:::get_covariates(model = gamgen_fit,
simulation_points = grilla_simulacion_centers,
seasonal_covariates,
simulation_dates,
control = control_sim)

aux <- gamwgen:::sf2raster(seasonal_covariates %>%
dplyr::filter(year == 2019, season == 1),
variable = 'seasonal_prcp') %>%
raster::as.data.frame(., xy = TRUE) %>%
dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
dplyr::mutate(., x = longitude, y = latitude) %>%
sf::st_as_sf(., coords = c('x', 'y')) %>%

```

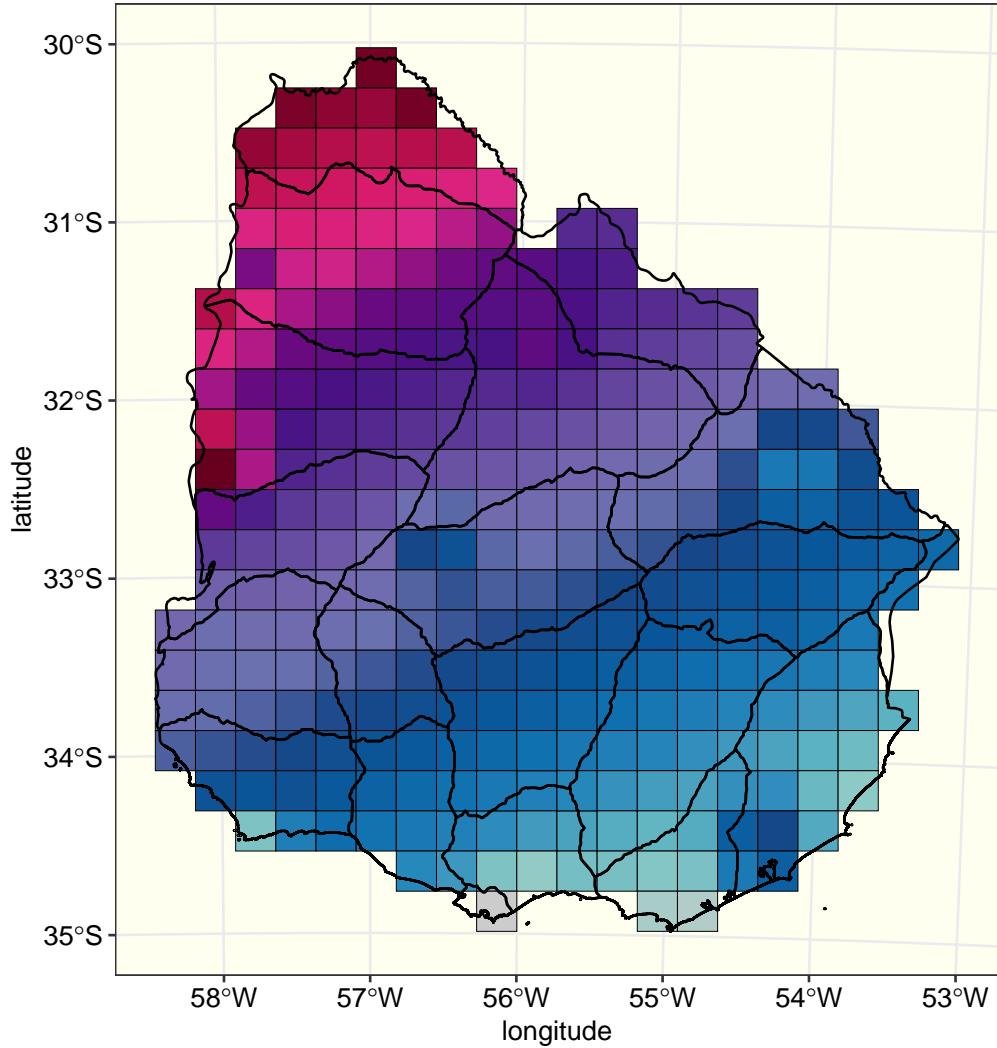
```
sf::st_set_crs(., 32721) %>%
sf::st_intersection(uruguay) %>%
dplyr::select(latitude, longitude, z, geometry)

colores <- colorRampPalette(c("#cccccc", "#7bcc4", "#2b8cbe", "#0868ac",
 "#084081", "#807dba", "#6a51a3", "#54278f", "#3f007d", "#e6338a",
 "#ce1256", "#67001f"))

quiebres <- c(0.0, 5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 110.0)

etiquetas <- c('0', '5', '10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '110')

ggplot(data = aux,
ggplot2::aes(x = longitude, y = latitude, fill = z)) +
geom_tile(colour="black") +
scale_fill_gradientn(name = "Precipitation [mm]",
colours = colores(17),
breaks = quiebres,
labels = etiquetas) +
ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
ggplot2::theme_bw() +
ggplot2::theme(panel.background = element_rect(fill = "ivory"),
axis.text = element_text(size = 11, colour = 1),
legend.key.height = unit(2.5, "cm"))
```



1199

```

aux <- gamwgen:::sf2raster(seasonal_covariates %>%
  dplyr::filter(year == 2019, season == 1),
  variable = 'seasonal_tmax') %>%
  raster::as.data.frame(., xy = TRUE) %>%
  dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
  dplyr::mutate(., x = longitude, y = latitude) %>%
  sf::st_as_sf(., coords = c('x', 'y')) %>%
  sf::st_set_crs(., 32721) %>%
  sf::st_intersection(uruguay) %>%

```

```

dplyr::select(latitude, longitude, z, geometry)

1200 ## Warning: attribute variables are assumed to be spatially constant throughout all
1201 ## geometries

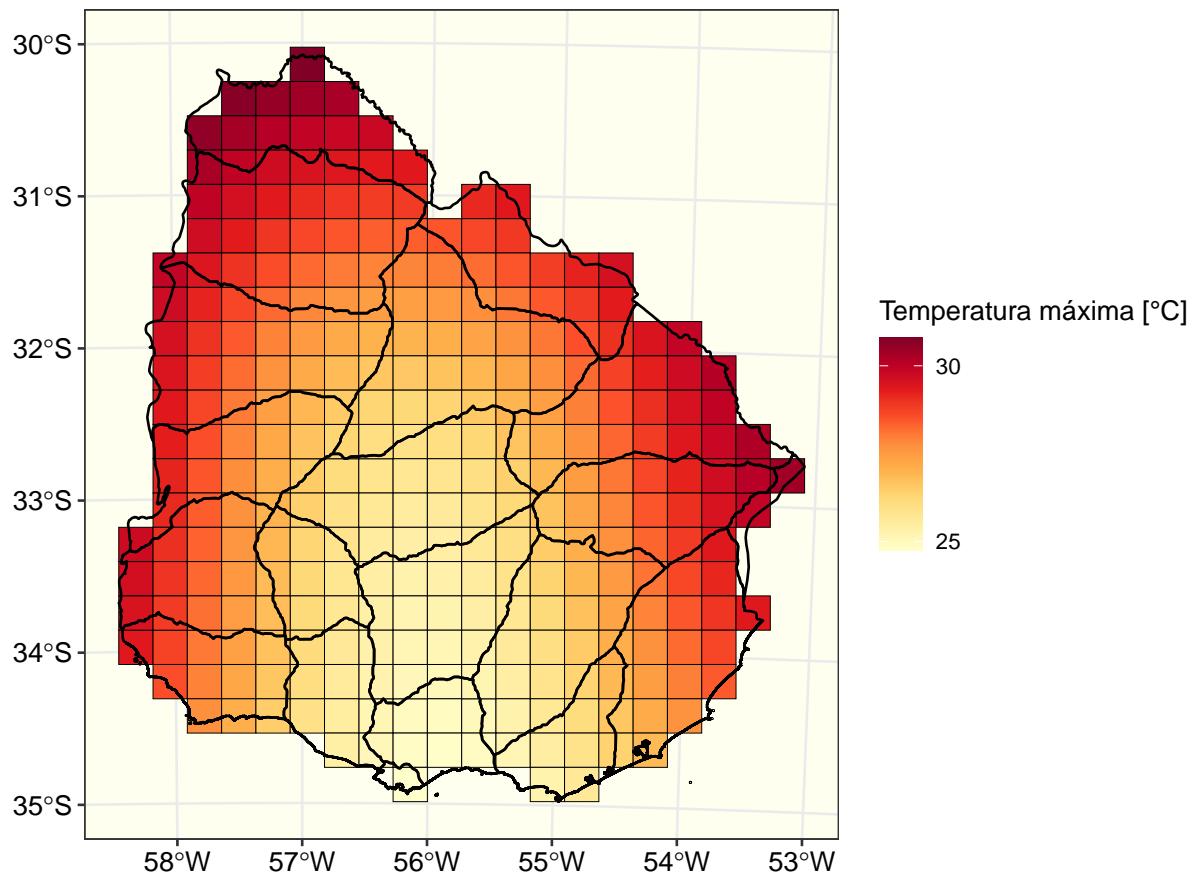
colr <- colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))

quiebres <- c(0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0)

etiquetas <- c('0', '5', '10', '15', '20', '25', '30', '35', '40', '45')

ggplot(data = aux,
        ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  geom_tile(colour="black") +
  scale_fill_gradientn(name = "Temperatura máxima [°C]",
                        colours = colr(9),
                        breaks = quiebres,
                        labels = etiquetas) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                axis.text = element_text(size = 11, colour = 1))

```



1202

1203 Luego se procede a la simulación de datos meteorológicos. Los argumentos de la función de  
1204 simulación son:

- 1205 • **model**: objeto con el resultado de la función `local_calibrate()`
- 1206 • **simulation\_locations**: objeto tipo `sf` con la ubicación de las estaciones a simu-  
1207 lar. Las estaciones usadas deben haber sido incluidas en el proceso de ajuste. No es  
1208 necesario que todas estén presentes, se pueden generar series solo sobre algunas de  
1209 ellas.
- 1210 • **start\_date**: fecha de comienzo de la generación de series sintéticas. Si no se incluyeron  
1211 covariables estacionales en el ajuste, la fecha de comienzo es completamente arbitraria.

1212 Caso contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de  
 1213 covariables, ni tampoco posterior. Se debe introducir una fecha en formato **date**  
 1214 • **end\_date**: fecha de fin de la generación de series sintéticas. Si no se incluyeron co-  
 1215 variables estacionales en el ajuste, la fecha de fin es completamente arbitraria. Caso  
 1216 contrario, la fecha de comienzo no puede ser anterior al inicio de la serie de covariables,  
 1217 tampoco puede ser posterior. Se debe introducir una fecha en formato **date**  
 1218 • **control**: objeto de control creado con la función **control\_sim()**.  
 1219 • **output\_folder**: ruta al directorio donde se guardarán los resultados, tanto finales  
 1220 como intermedios.  
 1221 • **output\_filename**: nombre del archivo de salida. Para facilitar la interoperabilidad,  
 1222 el archivo generado es un archivo de texto en formato separado por comas (.csv)  
 1223 • **seasonal\_covariates**: datos agregados trimestrales. Si el ajuste se realizó con co-  
 1224 variables, la generación también debe realizarse con ellas. Caso contrario se producirá  
 1225 un error. Se debe introducir un data frame con los valores agregados para las tres  
 1226 variables (precipitación y temperaturas máxima y mínima) pero no necesariamente  
 1227 deben ser los mismos a los utilizados en el ajuste. Si se desean simular tendencias de  
 1228 algún tipo, ya sea de un modelo de cambio climático o arbitrarias, se deben perturbar  
 1229 estas variables trimestrales e introducirlas aquí. Estas series si deben tener la misma  
 1230 longitud que el período a generar  
 1231 • **verbose**: controla la impresión de mensajes en la consola. FALSE por defecto.

```

# Al correr la función se realiza la generación de series para cada una de las estacio-
# En este caso, por cuestiones de tiempo, vamos a cargar un objeto con los resultados
simulated_climate <- gamwgen::spatial_simulation(model = gamgen_fit, # Objeto con los r
  simulation_locations = grilla_simulacion_centers, # Estaciones para las cuales simu-
  start_date = as.Date('2019-01-01'), # Fecha de comienzo de las simulaciones
  end_date = as.Date('2019-01-31'), # Fecha de fin de las simulaciones
  control = control_sim, # Objeto con la configuración

```

```

    output_folder = getwd(), # Directorio donde se guardarán los resultados
    output_filename = 'simulations.csv', # Nombre del archivo de salida
    seasonal_covariates = seasonal_covariates, # Covariables estacionales
    verbose = FALSE) # Impresión de mensajes en la consola

```

1232 Esta función produce dos tipos de resultados: una lista que permanece en el ambiente de R y  
 1233 los datos generados que son guardados como .csv en el directorio indicado precedentemente.

```

# Copiamos el archivo preajustado a nuestro directorio de trabajo

if (!fs::file_exists('output_data/spatial/simulated_spatial_conditional.RData')) {
  fs::file_copy(system.file('/autorun/spatial', "simulated_spatial_conditional.RData",
                           new_path = 'output_data/spatial/simulated_spatial_conditional.RData')
}

# Cargamos el archivo recientemente creado

load('output_data/spatial/simulated_spatial_conditional.RData')

# Clase del objeto con el ajuste del generador

class(simulated_climate)

1234 ## [1] "list"           "gamwgen.climate"

# Contenido del modelo

names(simulated_climate)

1235 ## [1] "nsim"
1236 ## [2] "seed"
1237 ## [3] "realizations_seeds"
1238 ## [4] "simulation_points"
1239 ## [5] "output_file_with_results"

```

```
1240 ## [6] "output_file_fomart"  
1241 ## [7] "rdata_file_with_fitted_stations_and_climate"  
1242 ## [8] "exec_times"
```

1243 La lista contiene los siguientes objetos:

- 1244 • `nsim`: cantidad de realizaciones.
- 1245 • `seed`: semilla general para toda la generación. Corresponde a la que se incluye en la  
1246 función de control.
- 1247 • `realization_seeds`: semillas para cada una de las realizaciones. Esto permite replicar  
1248 los resultados.
- 1249 • `simulation_points`: puntos donde se generaron las series sintéticas.
- 1250 • `output_file_with_results`: nombre del archivo con los resultados.
- 1251 • `output_file_format`: tipo de archivo de salida, en este caso `.csv`.
- 1252 • `rdata_file_with_fitted_stations_and_climate`: archivo `.RData` con los datos me-  
1253 teorológicos observados que fueron utilizados en el ajuste. También se incluyen los  
1254 metadatos de cada uno de esos puntos.
- 1255 • `exec_times`: tiempo de ejecución de la generación.

1256 Ahora veremos el formato del archivo de salida que contiene las series sintéticas.

1257 Para desmenuzar la generación del tiempo local se pueden correr algunas de las funciones  
1258 que forman parte de `local_simulation`. Por ejemplo, del objeto `gamgen_fit` se extraen  
1259 los residuos y con la función `gamwgen:::generate_residuals_statistics` se calculan los  
1260 parámetros.

1261 En la tabla anterior se muestran los parámetros necesarios para generar el tiempo local de  
1262 las temperaturas máxima y mínima.

```
gen_noise_params <- gamwgen:::generate_month_params(  
  residuals = gamgen_fit$models_residuals,
```

```

observed_climate = gamgen_fit$models_data,
stations = gamgen_fit$stations)

# Ejemplo de variograma de residuos de temperatura máxima para días secos

gen_noise_params[[1]]$variogram_parameters$tmax_dry

1263 ## [1] 0.000000 7.045242 493.298928

1264 Explicar variograma.

1265 Los parámetros para cada uno de los meses permiten generar valores de tiempo local para
1266 las dos temperaturas a partir de una distribución normal multivariada.

1267 A continuación se muestra un ejemplo para el año 2019 sólo para los días lluviosos.

```

```
RandomFields::RFoptions(printlevel = 0, spConform = FALSE)
```

```

grilla_simulacion_centers <- uruguay %>%
  sf::st_transform(gamgen_fit$crs_used_to_fit) %>%
  sf::st_make_grid(cellsize = 25000, what = 'centers') %>%
  sf::st_sf(grid_id = 1:length(.), geometry = .) %>%
  sf::st_intersection(uruguay) %>%
  dplyr::select(geometry) %>%
  dplyr::mutate(latitude = sf::st_coordinates(.)[,2],
    longitude = sf::st_coordinates(.)[,1])

```

```
# Mapa de Uruguay
```

```
uruguay <- raster::getData('GADM', country='URY', level=1) %>%
  sf::st_as_sf(.) %>%
```

```

sf::st_transform(crs = 32721)

temp_local <- control_sim$temperature_noise_generating_function(
  simulation_points = grilla_simulacion_centers,
  gen_noise_params = gen_noise_params,
  month_number = 1L,
  selector = 'Dry',
  seed = 1234) %>%
  dplyr::mutate(date = as.Date(paste0('2019-', '01', '-01'))) %>%
  dplyr::filter(date == as.Date('2019-01-01')) %>%
  tidyrr::gather(variable, valor, -c('geometry'))

tmax_residuals <- gamwgen:::sf2raster(temp_local %>%
  dplyr::filter(variable == 'tmax_residuals'),
  'valor') %>%
  raster::as.data.frame(., xy = TRUE) %>%
  dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%
  dplyr::mutate(., x = longitude, y = latitude) %>%
  sf::st_as_sf(., coords = c('x', 'y')) %>%
  sf::st_set_crs(., 32721) %>%
  sf::st_intersection(uruguay) %>%
  dplyr::select(latitude, longitude, z, geometry)

colr <- colorRampPalette(RColorBrewer::brewer.pal(9, 'YlOrRd'))

quiebres <- c(0.0, 5.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0)

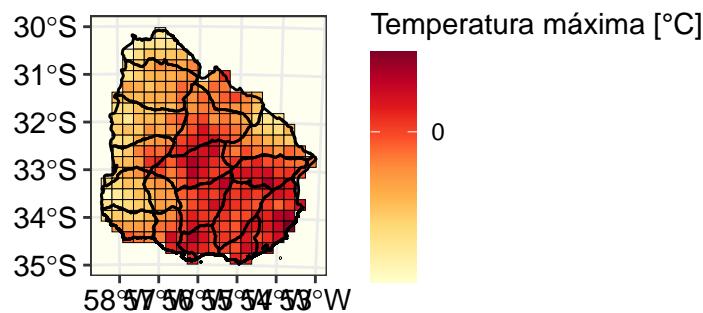
```

```

etiquetas <- c('0', '5', '10', '15', '20', '25', '30', '35', '40', '45')

ggplot(data = tmax_residuals,
       ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  geom_tile(colour="black") +
  scale_fill_gradientn(name = "Temperatura máxima [°C]",
                        colours = colr(9),
                        breaks = quiebres,
                        labels = etiquetas) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE) +
  ggplot2::theme_bw() +
  ggplot2:: labs(title = "", x = "", y = "") +
  ggplot2::theme(panel.background = element_rect(fill = "ivory"),
                 axis.text = element_text(size = 11, colour = 1))

```



1268

1269 La línea naranja corresponde a la serie para temperaturas máximas y la azul para temper-  
1270 aturas mínimas.

```
# Se carga el set de datos simulados  
  
simulated_climate <- readr::read_csv(here::here('output_data/spatial/simulated_spatial_0'))  
  
# Primeras filas del objeto de salidas  
  
knitr::kable(simulated_climate[1:10,])
```

realization	point_id	longitude	latitude	date	tmax	tmin	prcp
1	1	453759.4	6590418	2019-01-01	33.03589	19.72844	1.826571
1	2	553759.4	6590418	2019-01-01	31.77257	19.65962	27.883864
1	3	578759.4	6590418	2019-01-01	32.71455	20.47479	32.879767
1	4	428759.4	6615418	2019-01-01	30.64363	16.02136	7.032041
1	5	453759.4	6615418	2019-01-01	31.83107	18.12258	14.671730
1	6	478759.4	6615418	2019-01-01	33.32291	18.75998	14.781878
1	7	503759.4	6615418	2019-01-01	32.22893	18.78158	11.007127
1	8	528759.4	6615418	2019-01-01	31.95957	17.71497	24.732878
1	9	553759.4	6615418	2019-01-01	31.93247	17.65988	26.055372
1	10	453759.4	6640418	2019-01-01	30.67991	17.13227	16.806296

1272 El resultado de la generación es un archivo .csv que contiene la siguiente información:

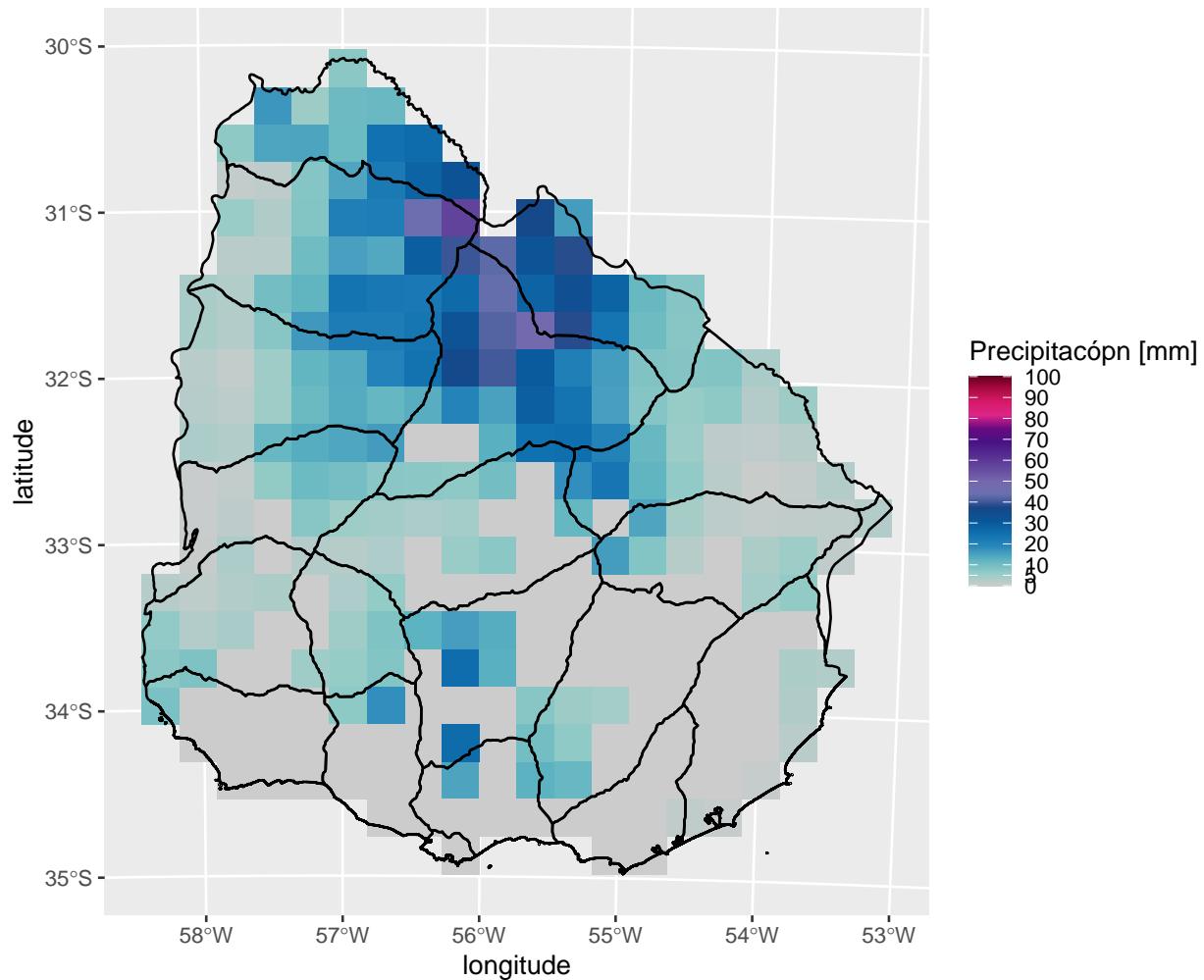
- **realization**: número de realización. Es un valor entero entre 1 y la cantidad de realizaciones definida por el usuario.
  - **station\_id**: número único de identificación de la estación meteorológica o del punto arbitrario.
  - **date**: fechas de cada uno de los días de la simulación.
  - **tmax**: valores de temperatura máxima generada expresada en °C.
  - **tmin**: valores de temperatura mínima generada expresada en °C.
  - **prcp**: valores de precipitación diaria generada expresada en mm.

1281 La siguiente Figura muestra un ejemplo de las series de temperaturas máximas y mínimas  
1282 generadas.

```
uruguay <- raster::getData('GADM', country='URY', level=1) %>%  
sf::st_as_sf(.) %>%  
sf::st_transform(crs = 32721)  
  
simulated_climate_geo <- simulated_climate %>%  
dplyr::filter(date == as.Date('2019-01-01'),  
realization == 1) %>%  
sf::st_as_sf(coords = c('longitude', 'latitude'), crs = 32721) %>%  
gamwgen:::sf2raster(., 'prcp') %>%  
raster::as.data.frame(., xy = TRUE) %>%  
dplyr::rename(., 'longitude' = 'x', 'latitude' = 'y') %>%  
dplyr::mutate(., x = longitude, y = latitude) %>%  
sf::st_as_sf(., coords = c('x', 'y')) %>%  
sf::st_set_crs(., 32721) %>%  
sf::st_intersection(uruguay) %>%  
dplyr::select(latitude, longitude, z, geometry)  
  
1283 ## Warning: attribute variables are assumed to be spatially constant throughout all  
1284 ## geometries  
  
colores <- colorRampPalette(c("#cccccc", "#7bccc4", "#2b8cbe", "#0868ac",  
"#084081", "#807dba", "#6a51a3", "#54278f", "#3f007d", "#e74c3c",  
"#ce1256", "#67001f"))  
  
quiebres <- c(0.0, 5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0, 110.0)
```

```
etiquetas <- c('0', '5', '10', '20', '30', '40', '50', '60', '70', '80', '90', '100', '105', '110', '115', '120', '125', '130', '135', '140', '145', '150', '155', '160', '165', '170', '175', '180', '185', '190', '195', '200', '205', '210', '215', '220', '225', '230', '235', '240', '245', '250', '255', '260', '265', '270', '275', '280', '285', '290', '295', '300', '305', '310', '315', '320', '325', '330', '335', '340', '345', '350', '355', '360', '365', '370', '375', '380', '385', '390', '395', '400', '405', '410', '415', '420', '425', '430', '435', '440', '445', '450', '455', '460', '465', '470', '475', '480', '485', '490', '495', '500', '505', '510', '515', '520', '525', '530', '535', '540', '545', '550', '555', '560', '565', '570', '575', '580', '585', '590', '595', '600', '605', '610', '615', '620', '625', '630', '635', '640', '645', '650', '655', '660', '665', '670', '675', '680', '685', '690', '695', '700', '705', '710', '715', '720', '725', '730', '735', '740', '745', '750', '755', '760', '765', '770', '775', '780', '785', '790', '795', '800', '805', '810', '815', '820', '825', '830', '835', '840', '845', '850', '855', '860', '865', '870', '875', '880', '885', '890', '895', '900', '905', '910', '915', '920', '925', '930', '935', '940', '945', '950', '955', '960', '965', '970', '975', '980', '985', '990', '995', '1000')

ggplot2::ggplot(data = simulated_climate_geo) +
  ggplot2::geom_tile(ggplot2::aes(x = longitude, y = latitude, fill = z)) +
  ggplot2::scale_fill_gradientn(name = "Precipitacópn [mm]", 
    colours = colores(17),
    breaks = quiebres,
    labels = etiquetas,
    limits = c(0, 100)) +
  ggplot2::geom_sf(data = uruguay, fill = NA, color = "black", inherit.aes = FALSE)
```



1285

```
#cowplot::ggdraw() +
#  cowplot::draw_plot(tmax_plot, x = 0, y = 0.5, width = 1, height = .5) +
#  cowplot::draw_plot(tmin_plot, x = 0, y = 0, width = 1, height = .5)
```

1286 En el panel superior se muestra la temperatura máxima observada (negra) y las temperaturas  
 1287 sintéticas (naranja) para el año 2019. En el inferior se muestra la temperatura mínima diaria  
 1288 observada (negra) y cada una de las realizaciones (azul).

1289 **References**

- 1290 Kleiber, W., Katz, R.W., Rajagopalan, B., 2012. Daily spatiotemporal precipitation simu-  
1291 lation using latent and transformed gaussian processes. Water Resources Research 48.
- 1292 Kleiber, W., Katz, R.W., Rajagopalan, B., 2013. Daily minimum and maximum temperature  
1293 simulation over complex terrain. Ann. Appl. Stat. 7, 588–612.
- 1294 Simpson, G., 2018. Introducing *gratia*. From the Bottom of the Heap.
- 1295 Verdin, A., 2016. Stochastic space-time modeling for agricultural decision support in the  
1296 argentine pampas (Thesis).
- 1297 Wickham, H., 2011. Ggplot2. Wiley Interdisciplinary Reviews: Computational Statistics 3,  
1298 180–185.
- 1299 Wood, S.N., 2001. Mgcv: GAMs and generalized ridge regression for r. R news 1, 20–25.
- 1300 Wood, S.N., 2017. Generalized additive models: An introduction with r. Chapman;  
1301 Hall/CRC.